INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600
Bidirectional Ring: An Interconnection Network for Shared-Memory Multiprocessor Systems

by

Muhammad Jaseemuddin

A thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy
Graduate Department of Electrical and Computer Engineering
University of Toronto

© Copyright by Muhammad Jaseemuddin 1997
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

0-612-41554-6
Abstract

Bidirectional Ring: An Interconnection Network for Shared-Memory Multiprocessor Systems

Muhammad Jaseemuddin
Doctor of Philosophy
Graduate Department of Electrical and Computer Engineering
University of Toronto
1997

A bidirectional ring is proposed in this thesis as an interconnection network for shared-memory multiprocessors to improve their performance and reliability. It consists of two rings transmitting packets in opposite directions. The constituent rings are bit-parallel and use the slotted access protocol. The bidirectional ring allows a request packet to traverse one ring from a source to the destination, and the response packet to return on the other ring; hence, a request and response transaction does not require traversal of the entire ring. This transmission protocol is advantageous in exploiting the communication locality of application programs to improve their performance. The existence of two rings in the bidirectional ring improves the reliability of the system.

A cache coherence protocol is proposed for shared-memory multiprocessors using the bidirectional ring interconnect. The protocol ensures sequential consistency. A reliable implementation of this protocol is also given that can handle node and link failures. The fault handling logic is concentrated in the bypass path, which is activated only after detecting a fault, hence it does not affect the normal operation of the ring.

The performance of the bidirectional ring is evaluated in comparison with a hierarchy of unidirectional rings. The simulation is performed using system parameters analogous to the NUMAchine prototype. A set of applications from the SPLASH-2 benchmark suite is used for this study. The bidirectional ring shows up to 50% lower latency than the hierarchy. Its performance is less sensitive to communication patterns. Comparing the overall performance with the hierarchy, the program execution time is reduced up to 30% on the bidirectional ring. The bidirectional ring also shows a tendency to balance the network load. The utilizations of the two constituent rings has been found to be within 5% of each other. The degradation in the performance of a faulty bidirectional ring is relatively low; a maximum increase of 6%
the execution time is found for all the benchmark applications we have simulated. The faulty ring shows the same load balancing characteristic of a good ring.
Acknowledgements

My foremost gratitude is to God who created me, provides sustenance for me, endowed me with intelligence and patience, necessary for this work, and to whom would I return.

I can find no word to thank my supervisor, Prof. Zvonko Vranesic, without whose guidance and support I cannot imagine doing this work. I am also grateful to professors P. Chow, V. C. Hamacher, K. Sevcik, E. Sousa, and M. Stumm, who put their time and effort in reading my thesis and providing me invaluable suggestions that have improved the presentation of this thesis.

I am indebted to my parents for their continuous support throughout my life, especially their endurance of living without me for so many years. I am grateful to my wife Fauzia, who has been the only source of comfort and cheer in the last stressful days of my work. I can never equate her love and support. I am also grateful to my brother and sisters and their families for their prayers.

I am thankful to Robin Grindley, who allowed me to modify and use his simulator. I would also like to thank G. Ravindran and D. Kulkarni, who advised me on many occasions. Kulki needs special mention for allowing me to use his latex style file. I would also like to acknowledge the information Alex Grbic and Kelvin Loveless have given me in response to my questions. My colleagues in SF2206 deserve my thanks too for providing a congenial environment. Sarah Cherian, Ella Lund-Thomsen, and Kelly Chan deserve my special thanks for giving me administrative help during the course of this work. My special thanks to Mazen Saghir, Halim Yanikomeroglu, Amer Shalaby, Yasser Rasheed, Mohammad Khalid, Saiful Islam, and other friends, who provided with invaluable company and friendship over the long period of five years, helped shape my personality, and enhanced my understanding of life.

Finally, I would like thank Quaid-e-Azam Aligarh Scholarship Trust, ITRC and the University of Toronto for their financial support.
## Contents

1 Introduction ....................................................... 1

2 Related Work .......................................................... 5

  2.1 Single Unidirectional Rings ........................................ 6
  2.1.1 ExpressRing .................................................. 6
  2.1.2 Scalable Coherent Interface (SCI) ............................ 8

  2.2 Hierarchy of Unidirectional Rings ................................ 12
  2.2.1 Kendall Square Research’s KSR-1 ............................ 14
  2.2.2 NUMAchine .................................................. 16

  2.3 Bidirectional Rings ............................................... 19
  2.3.1 MetaRing .................................................... 19
  2.3.2 GigaRing .................................................... 20

3 Bidirectional Ring ................................................... 22

  3.1 Routing .......................................................... 24

4 Preliminary Evaluation ............................................... 26

  4.1 System Model .................................................... 26
  4.1.1 Simulation Methodology ....................................... 27

  4.2 Request-Response Protocol ....................................... 28

  4.3 Workload Models ................................................ 29

  4.4 Comparing Performance of the Bidirectional Ring and the Hierarchy of Unidirectional Rings ........................................ 30
  4.4.1 Network Performance .......................................... 31
  4.4.2 Uniform Workload ............................................ 31
  4.4.3 Bursty and Mirror Workloads. ................................ 33
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Configurations of different sizes</td>
<td>30</td>
</tr>
<tr>
<td>5.1</td>
<td>Some important request/response transactions</td>
<td>40</td>
</tr>
<tr>
<td>6.1</td>
<td>Base configuration</td>
<td>51</td>
</tr>
<tr>
<td>6.2</td>
<td>Benchmarks</td>
<td>53</td>
</tr>
<tr>
<td>6.3</td>
<td>Statistics of affected shared reads</td>
<td>63</td>
</tr>
<tr>
<td>6.4</td>
<td>Parallel execution times (in ms)</td>
<td>70</td>
</tr>
</tbody>
</table>
List of Figures

2.1 SCI pointer operations ................................................................. 9
2.2 Lynx block diagram ................................................................. 11
2.3 An example of the hierarchy of unidirectional rings ...................... 13
2.4 The routing mask ................................................................. 17
3.1 An example of the bidirectional ring ........................................... 22
3.2 Routing scheme for the bidirectional ring .................................... 25
4.1 Latency for uniform traffic and CLM1 ........................................ 32
4.2 Network utilization for a 256 processor system running uniform workload 33
4.3 Latency for uniform traffic and CLM2 ........................................ 34
4.4 Latency for bursty traffic and CLM1 ........................................ 34
4.5 Latency for mirror traffic and CLM1 ........................................ 35
4.6 Latency for hotspot traffic and CLM1 ........................................ 36
5.1 Algorithm for processing exclusive-read request .......................... 42
5.2 Algorithm for processing upgrade request ................................... 43
5.3 State diagram for enforcing the sequential consistency .................. 44
5.4 Algorithm for processing shared-read request ................................ 45
5.5 Algorithm for multicasting invalidation on the bidirectional ring .... 47
5.6 An example of invalidation packet flow ....................................... 48
6.1 A Station ............................................................................. 51
6.2 Secondary cache miss profile ................................................... 59
6.3 Transaction latency vs. line size for FFT .................................... 61
6.4 Transaction latency vs. line size for Radix .................................. 61
6.5 Transaction latency vs. line size for Ocean .................................. 62
6.6 Network utilization vs. line size for FFT .................................................. 63
6.7 Network utilization vs. line size for Radix .................................................. 64
6.8 Network utilization vs. line size for Ocean .................................................. 64
6.9 Average global ring access delay (in ring cycles) ......................................... 66
6.10 Average local ring access delay (in ring cycles) .......................................... 66
6.11 Average ring-0 access delay (in ring cycles) ............................................. 67
6.12 Average ring-1 access delay (in ring cycles) ............................................. 67
6.13 Speedup vs. system size (in no. of processors) for Ocean and Water .............. 68
6.14 Speedup vs. system size (in no. of processors) for FFT and Radix ................. 69
6.15 Speedup vs. system size (in no. of processors) for Cholesky and Ray trace ...... 69
6.16 Normalized execution time for the hierarchy and the bidirectional ring running at different clock speed on 64-processor system ................................................. 71

7.1 An example of ring-down mode of failure .................................................... 74
7.2 An example of bypass mode of failure ......................................................... 75
7.3 Network interface to perform bypassing ....................................................... 76
7.4 Algorithm for multicasting invalidation in the bypass mode ............................ 79
7.5 An example of invalidation packet flow in a good network ............................... 80
7.6 An example of invalidation packet flow in a faulty network ............................ 81
7.7 Network utilization of good and faulty bidirectional rings for FFT .................. 84
7.8 Network utilization of good and faulty bidirectional rings for Ocean ............... 84
7.9 Network utilization of good and faulty bidirectional rings for Radix ............... 85
7.10 Network utilization of good and faulty bidirectional rings for Ray trace .......... 85
7.11 Network utilization of good and faulty bidirectional rings for Water-Nsquared ... 86
7.12 Network utilization of good and faulty bidirectional rings for Cholesky .......... 86
7.13 Normalized execution time for good and faulty bidirectional rings ............... 87
Interconnection networks play a vital role in determining the overall performance of multiprocessor systems. Even though enhanced understanding of parallelization of sequential programs has improved the programmer's capability of writing parallel code that is optimized for load balancing, locality, and parallelization and synchronization overhead, the communication latency still contributes a significant portion to the execution time. Therefore, the interconnection network has been a very active and prolific area of research. Numerous topologies and protocols have been proposed. It is clear that there are important advantages and disadvantages in various schemes. For small multiprocessors (up to 16 processors) it is difficult to find a scheme that is more useful than a simple single-bus interconnection. Very large systems (on the order of thousands of processors) are of diminishing practical interest, because the market for such machines is small. Thus, it is the medium-sized systems that warrant careful investigation; these systems are likely to range from 16 to 512 processors. There are several reasons why the interconnection network for this system size needs careful attention.

First, the use of multiprocessors in the commercial market place will expand rapidly. The emerging applications include transaction processing, decision support systems, mission critical systems, and multimedia systems. These applications demand the system to provide high reliability and availability along with high speed.

Second, compute servers are increasingly used in multiprogramming environments. Multiprogramming requires high throughput from the system, which means the interconnection network must provide high bandwidth.

Third, the commercial market is very cost sensitive. The success of systems in this market depends upon their cost-performance characteristic. The issue of cost demands that all components of the system, including the interconnection network, be cost-effective and simple.

Ring-based interconnection networks offer several important advantages. They allow simple point-to-point connections that can be controlled using simple routing protocols, resulting in minimal packaging constraints. They allow simple network interfaces, wide bit-parallel trans-
mission paths, and high clock speed. They provide a natural multicast mechanism that is useful for the distribution of both data and control information. They facilitate the implementation of simple cache coherence protocols, if the structure ensures that there is a unique path between any two nodes so that the ordering of packets is always maintained.

These features of ring-based networks have attracted the attention of designers of multiprocessor systems. The SCI standard of IEEE [14] recommends using the ring topology to connect a large number of processors, using a linked-list based cache coherence protocol. Sequent's STiNG [32] and HP-Convex' Exemplar [50] are two commercial implementations of SCI. Barroso and Dubois proposed a single unidirectional ring, called ExpressRing, and a snooping cache coherence scheme to build up to a 64-processor shared-memory machine [1]. They claim that the Express Ring performs better than a 64-bit split-transaction pipelined bus [2].

A large single ring cannot be used to connect hundreds of processors. The communication latency on a single ring is independent of the distance between the source and destination, if each request and response transaction requires traversal of the entire ring. This makes a single ring incapable of exploiting communication locality shown by a large number of parallel applications. Another drawback of the single ring is its low reliability, as it suffers from a single point of failure.

A hierarchy of smaller rings is attractive for exploiting communication locality and achieving scalability. For example, a two-level hierarchy can be constructed by connecting a number of small low-level rings, called local rings, through a high-level ring, called a global ring. Each local ring forms a cluster of processors whose size is fixed by the architecture. The hierarchy has been effectively used to implement multiprocessor systems. It was used in the KSR-1 [4] commercial system and several research prototypes, such as Hector [56] and NUMAchine [57]. The hierarchy is particularly effective in exploiting the communication locality that is concentrated within clusters. It is less effective when there exists a low degree of communication locality where each processor communicates frequently with processors that are not necessarily a part of the same cluster. In general, the local rings and the global ring in the hierarchy show different network loads. For applications that show all-to-all communication traffic, or low communication locality, the higher-loaded global ring affects the communication latency adversely. Although the hierarchy improves reliability, it still faces the problem of a single point of failure. The hierarchy may survive a fault in a local ring, but any fault in the global ring may cause network partitioning.

These three factors - static clustering, load unbalancing, and low reliability - lead to the design of a bidirectional ring. This thesis suggests a bidirectional ring as an alternative to the hierarchy of unidirectional rings to connect up to 256 processors in a multiprocessor system.
It allows dynamic formation of clusters of processors to exploit communication locality more efficiently. It also provides automatic load balancing on the communication links, and has potential for increased reliability.

To show the relative merits of the bidirectional ring, its various aspects are compared against a hierarchy of unidirectional rings. This thesis does not attempt to compare the bidirectional ring with other network structures, such as meshes. The debate about relative merits of rings, meshes, and other structures is still unsettled, though some comparisons are reported elsewhere [15, 39].

The thesis is organized as follows.

- Chapter 2 describes the related work.
- Chapter 3 introduces the bidirectional ring network. A routing scheme is also proposed which employs the routing protocol for the hierarchy by simulating different levels of rings as routing clusters.
- In Chapter 4 the results of a preliminary evaluation of the relative performance of the bidirectional ring and the hierarchy are presented. The performance is evaluated by running synthetic workloads on a simulated system. The synthetic workloads are used to study the network characteristics under different traffic pattern and load conditions, which is nearly impossible to achieve using real benchmarks.
- A cache coherence protocol for the bidirectional ring network is described in Chapter 5. The proposed protocol is an invalidation-based write-back protocol that is implemented by multicasting invalidation packets over the ring. An implementation scheme is proposed in Section 5.3.
- The performance of a bidirectional ring-based multiprocessor system is evaluated in relation to a system based on the hierarchy of unidirectional rings in Chapter 6. The performance data is collected by running a number of Splash-2 benchmark applications on a cycle-level detailed simulator of the two systems. The set of benchmark applications is carefully selected to represent a variety of communication patterns as well as to generate enough communication so that the impact of the network on the overall system performance is reflected.
- The reliability issues of the bidirectional ring are discussed in Chapter 7. Two modes of failure – ring down and bypass – are described in Section 7.1. Section 7.3 describes the implementation of the cache coherence protocol tolerating the bypass mode of failure.
Chapter 8 gives a summary and highlights the main achievements of the thesis. Some future work is also proposed.
The simplicity of unidirectional rings, which allow simple network interfaces, wide bit-parallel transmission paths, and high clock speed, has attracted the attention of designers of multiprocessor systems. The SCI standard of IEEE [14] proposes the ring topology to connect a large number of processors, using a linked-list based cache coherence protocol. Since a single long ring is likely to increase the latency of packet transfers, a hierarchy of rings has found application in several machines, such as Hector [56], KSR-1 [4], and NUMAchine [57]. The hierarchy is mainly used to exploit communication locality and to achieve scalability. The bidirectional ring has been proposed for local area networks [5], and an I/O network [44]. Kee and Hariri presented communication algorithms to perform data distribution, data collection, and data broadcast functions over a bidirectional ring based message passing multicomputer [25]. They do not propose a bidirectional ring based multicomputer architecture; rather the algorithms are designed by using a model architecture, called a virtual parallel system, that can be mapped to any multicomputer by simulating a bidirectional ring structure on the underlying communication platform to execute the proposed algorithms. The key results of this thesis show the viability of a bidirectional ring as an interconnection network for shared-memory multiprocessor systems. Our proposal for the bidirectional ring and preliminary results were first published in EuroPar '95 [20]. More recently, Oi and Ranganathan have also considered the bidirectional ring for a shared-memory multiprocessor system, and have developed an analytical model to study the performance of the bidirectional ring [36, 37].

This chapter is organized into three sections. The first section contains a discussion on single unidirectional rings and describes the proposed architectures that use single rings as the interconnect. Section 2 describes the systems based on the hierarchy of unidirectional rings. In Section 3 the usage of bidirectional rings as local area networks and I/O networks is discussed.
2.1 Single Unidirectional Rings

Single unidirectional rings have been used to connect a small number of nodes in multiprocessor systems. In a ring-based network, a message is decomposed into packets, which are the units of transfer across the ring. Three different access control mechanisms for rings have been used: (i) token ring [51], (ii) slotted ring [18], and (iii) buffer insertion ring [40]. In the token ring, a token is passed around the ring, and the node that holds the token transmits all its packets before releasing the token. Thus, only one node can access the ring at a time. The token ring guarantees a fair share of bandwidth allocation, but it precludes concurrent transmission. In a slotted ring a number of parallel bits constituting a slot move from one node to another in one clock cycle. A slot is a container that holds a packet and is routed independent of another slot; hence the slot is composed of a header part, containing routing and other control information, and a data part. A busy-bit in the slot header indicates whether the slot contains a packet or it is empty. For the slotted ring, the terms packet and slot are synonyms. The buffer insertion ring is a type of slotted ring with an insertion buffer at each node. When a node is ready to transmit and finds the insertion buffer empty, it proceeds with transmitting its packet. Meanwhile, if a busy slot arrives at the node that is to be forwarded to the next node, it is stored in the insertion buffer. The capacity of the insertion buffer determines the number of packets that are allowed to be transmitted in a row before flushing out the buffer.

2.1.1 ExpressRing

Buses are widely used to implement small shared memory multiprocessor systems, because they provide uniform memory access and a simple broadcast-based snooping cache coherence mechanism. But, due to contention and some electrical limitations on a bus, the connectivity provided by a bus is limited to a small number of processors; for example, the number of processors connected via a bus is usually 4 to 16. ExpressRing is a single unidirectional slotted ring proposed to address this problem [1]. It keeps the simplicity and broadcast capability of a bus, but extends the connectivity to up to 64 processors. Each node on the ring consists of dual processors, a fraction of shared memory space, and a cache with dual tags. There are two memory banks at each node; one for even addresses and one for odd addresses. There are up to 32 nodes on a single ring. The number of slots is three times greater than the number of nodes in the ring. A group of eight slots forms a frame. The first two slots of a frame contain request messages, one for even memory addresses and the other for odd memory addresses. The next five slots contain a response message, which is one cache line long. The last slot is an extra slot for the transfer of interrupts and other control information. The ring is proposed to run at
500 MHz, which means that a slot moves between the nodes in 2 ns. The slots are organized into frames to facilitate the snooping cache coherence protocol over a high speed ring, which is described below. The reason for reserving slots for even and odd memory addresses in a frame is to avoid memory bank conflicts which may occur when both requests in the frame are destined to the same node.

A snooping cache coherence scheme is proposed for the ExpressRing [1]. In this scheme, a request slot is examined by every node. When a node finds the data for that request, it sets the acknowledgement bit in the following frame, which indicates that the data will be supplied in the first available response slot. The reason for delaying acknowledgement by one frame-time is to give the node enough time to perform the snooping action. The requester checks the acknowledgement bit in the frame that immediately follows its request frame. It waits for the data upon receiving the acknowledgement. Similarly, a request slot that contains an invalidation request is examined by every node, and all those nodes that have a copy of the memory block in their caches will invalidate their copies. A full-map directory protocol is also evaluated against the snooping protocol for the ExpressRing, but the directory protocol is found to show inferior performance [2]. In a full-map directory protocol, the memory is divided into cache-line-sized blocks. Every node maintains a directory for its memory module, which has an entry for each memory block. The directory entry keeps pointers to all caches that have a copy of the memory block. This node is called the home node for that memory block. The full-map directory protocol that is considered in this study employs multicasting of invalidations over the ring, which needs one traversal of the ring. The reason for poor performance of the directory protocol is that the home node is required to arbitrate all remote read or write operations to a memory block. Hence, most of the remote transactions require one traversal of the ring. The ExpressRing using the proposed snooping cache coherence scheme performs better than a 64-bit split-transaction pipelined bus [2]. The authors hoped to show that the snooping cache coherence protocol on the slotted ring can be used to implement a shared-memory multiprocessor.

There are at least two major problems with the ExpressRing. The given time window, 16 ns for a frame, is not enough for a node to receive the packet, perform the cache directory look-up, and prepare the response. A simple directory look-up requires a memory access that is likely to take more than 16 ns. Receiving a packet means writing into the receive buffer and reading it subsequently from the buffer for processing. Any effort to implement snooping slows down the ring significantly. Thus, a snooping scheme for a ring is not a practical proposition. The other problem is related to the concept of framing. The frames must be synchronized for correct reception and transmission, which requires accurate recognition and delimiting of frames. In
addition to this, the transmission protocol that requires a requester to check the next slot for acknowledgement, restricts that only the requester can remove the request slot.

2.1.2 Scalable Coherent Interface (SCI)

The Scalable Coherent Interface (SCI) is the IEEE PI596 standard for network interface (logical and physical level) and cache coherence protocol to implement a shared-memory multiprocessor [14]. The network interface specifies an input and output link for each node, thus forming a ring of nodes that can operate at 1Gbyte/sec. The main feature of the SCI standard is its unique distributed directory based cache coherence protocol that is scalable up to 64K nodes.

In SCI, a distributed sharing list is maintained by linking the caches that share a cache line in the form of a doubly linked list, as shown in Figure 2.1. The directory entry at the memory block contains a pointer to the head of the list. Every node of the list is a cache that keeps two pointers with its copy of the line, a forward pointer to its successor and a backward pointer to its predecessor cache in the list. There are three operations on the list defined in the standard: (i) the addition operation to add a node to the list, (ii) the deletion operation to remove a node from the list, and (iii) the reduction operation to reduce the list to a single node list by invalidating all caches other than the head node which becomes the single node of the list. The head of the list has both read and write privileges for the cache line. In Figure 2.1a, an example of the addition operation is shown. When CacheA wants to fetch a line, it sends a request to the memory module containing that line. The memory module modifies its head pointer to point to CacheA, and sends its old head pointer to CacheA. If the memory has valid data, it also sends the data to CacheA. CacheA subsequently sends a request to the head node, Cache1, to change its forward pointer to point CacheA, thus making CacheA the new head. The head also responds with the data if the memory block is invalid. The deletion operation, as shown in Figure 2.1b involves pointer updates at the two adjacent nodes of the deleted node. The deleted node first sends its forward pointer to its successor, and then its backward pointer to its predecessor. The order of updates is important to start removing from the tail side in case of concurrent deletions. The reduction is performed when the head node wants to invalidate all copies of the line to gain an exclusive privilege, as illustrated in Figure 2.1c. The head node sends an invalidation request to its successor, which acknowledges this by returning its forward pointer. The head node sends an invalidation request to the node using the pointer that was received as an acknowledgement of the previous invalidation. The node returns its forward pointer in its acknowledgement message. The invalidation continues in this manner until all the nodes are invalidated by the head node. The invalidation operation involves $O(n)$ exchanges of messages, where $n$ is the number of nodes in the sharing list.
Figure 2.1: SCI pointer operations
Although the goal of SCI is to achieve scalability up to thousands of processors, the linear complexity of the protocol limits its applicability. The pointer updates in deletion are also more expensive than in any other directory based scheme. Another major problem with the protocol is that invalidating a list of size \( n \) may take \( n \) traversals of the ring. Each traversal involves a request-response transaction. The two notable variations of SCI are GLOW and STEM, which are optimized for different types of target systems [24]. GLOW is intended to be used in a system that comprises many SCI rings connected through bridges. This type of system is commonly found in networks of workstations. The target systems for STEM are large scale tightly coupled multiprocessor systems, such as massively parallel multiprocessors.

In the following subsections two commercial systems that use the SCI standard – STiNG and Exemplar – are discussed.

**STiNG**

STiNG is a cache coherent non-uniform memory access (CC-NUMA) multiprocessor system designed and built by Sequent Computer Systems [32]. The architecture is targeted for the commercial database market. A number of nodes are connected through an SCI interconnect. Each node is a 4-processor Symmetric Multiprocessor (SMP), called a *Quad*. In each quad, four Pentium Pro processors, two PCI I/O slots, a local memory module, a network cache, and a network (ring) interface card are connected by a bus. The snooping protocol provides cache coherence on the bus. A directory based SCI scheme is used to maintain coherence of data across the ring. The main reason for using SCI is that off-the-shelf components can be used to build the interconnect.

The network cache and ring interface are housed on the same card, called *Lynx*. The block diagram of the Lynx card is shown in Figure 2.2. There are three basic components of the Lynx: (i) P6 bus interface, (ii) network cache, and (iii) ring interface. Two sets of cache directories are provided to facilitate the snooping protocol – one on the bus side and the other on the ring side. Each set contains a directory of states for the local memory blocks and a directory of tags for the network cache. The P6 bus interface controls the snooping protocol by maintaining the bus-side directory. Besides state bits, the bus-side directory keeps a header pointer to the linked list of caches. The ring interface is composed of the directory controller and the interconnect controller. The directory controller maintains the ring-side directory, and it is also responsible for generating packets for remote transactions. The ring-side directory stores two pointers per memory block, a forward pointer and a backward pointer, in addition to state bits. The directory controller has a protocol engine, which gives the flexibility of using different protocols without making hardware modifications. It is also connected to the interconnect controller,
which provides the link and packet level interface to the SCI ring and manages the reception and transmission of packets. The interconnect controller contains a GaAs chip developed by Vitesse Semiconductor Corporation. The chip provides the link and packet level protocol for an SCI network. The choice of this chip led the designers to use SCI despite its complexity and lower performance than other directory protocols. But, the flexibility of the protocol engine allows them to explore better protocols in the future to improve performance [32].

![Lynx block diagram](image)

**Figure 2.2: Lynx block diagram**

**Exemplar**

The Exemplar architecture is designed for HP-Convex S- and X-Class systems [6]. The architecture supports a shared-memory programming model by providing a single logical shared address space that is distributed across the physical memories. It supports up to 512 processors and 512 Gbytes of physical memory that can be connected through three levels of interconnect. Two processors are connected with their 1-Mbyte data caches to form a processing module. Each processing module also contains up to three I/O ports and a crossbar interface. The crossbar is used to interconnect up to eight processing modules with the memory subsystem which comprises eight memory controllers. Each of these controllers controls a four-way interleaved memory. A set of processing modules connected with the memory subsystem through a crossbar is called a *hypernode*. A hypernode is a Symmetric Multiprocessor (SMP) that uses a crossbar interconnect. Multiple SCI rings can be used to connect up to 32 hypernodes to implement a large system. The SCI cache coherence protocol is used to maintain coherence across hypernodes.
The Convex SPP-1000 is an implementation of the Exemplar architecture [50]. A notable change in the implementation is that the memory subsystem of a hypernode is distributed across the processing modules. Each processing module comprises a pair of HP PA7100 processors that run at 100 MHz clock and two memory modules of up to 16 Mbytes each. There is a separate 1 Mbyte data and instruction cache for each processor within the processing module. The cache line size is 32 bytes. A 5-port high speed crossbar is used to connect 4 processing modules and an I/O port of a hypernode. Four SCI rings are used to connect 16 hypernodes. The first processing modules of all hypernodes form one ring, the second modules of all hypernodes are connected through the second ring, and so on. When a processor of one hypernode needs access to a remote memory located in the processing module of another hypernode such that the processor and memory are not connected by the same SCI ring, the request is routed through the processing module of the source hypernode that is connected to the target ring. In this way, the request is first transmitted across the crossbar of the source hypernode to the intermediate processing module, and from there it traverses across the SCI ring to reach the target memory. Full SCI cache coherence protocol is used across the whole system. A portion of the physical memory is allocated as a cache buffer to store the remote data that is fetched from hypernodes of the same ring. Thus, a collection of all such cache buffers within a hypernode acts like a network cache.

2.2 Hierarchy of Unidirectional Rings

A single ring cannot be used to connect a large number of processors. The communication latency on a single ring is independent of the distance of communication, which makes the single ring incapable of exploiting communication locality exhibited by many parallel applications. A ring connecting more than eight nodes shows worse latency than a system connecting the same number of nodes in a hierarchy of smaller single rings [39]. A representative example of a hierarchy is shown in Figure 2.3, in which local rings connect a certain number of nodes and these rings are interconnected by a global ring. Rings in the hierarchy are bit-parallel, unidirectional, and use a slotted protocol. A message is divided into packets and one packet can be transported by a slot. An interface that connects two rings is called an inter-ring interface. An inter-ring interface is a simple switch with two buffers, an up-buffer that is used to store the packets going from the local to the global ring, and a down-buffer that is used to move the packets from the global to the local ring. In a typical shared-memory system implemented using the hierarchy of unidirectional rings, a node consists of one or more processors, a memory module (holding a portion of the shared address space), I/O circuitry, and a communication switch that connects
it to the local ring. Each processor is assumed to have its own primary and secondary caches. Larger hierarchies are implemented by increasing the number of levels in the system.

![Diagram of hierarchy of unidirectional rings]

Figure 2.3: An example of the hierarchy of unidirectional rings

Each local ring in the hierarchy forms a cluster of processors. Thus, the size of a cluster is fixed. The static nature of the cluster causes two problems. First, it favors the assignment of processors to a process to suit the cluster size. Second, the performance of the network becomes very sensitive to the memory access patterns that exhibit variable amounts of inter-cluster communication. A process that spawns as many threads as the number of processors in a cluster can naturally fit in one cluster, but the addition of one more thread causes it to span two local rings. This introduces a communication overhead proportional to the size of the second-level ring connecting the two local rings. We call this the spanning overhead. If it takes more levels for a packet to cross to reach the destination ring, then each level will add an overhead proportional to the size of the ring at that level. Furthermore, when a packet ascends or descends the hierarchy it passes through several inter-ring interface buffers, each of which contributes a variable amount of time to the overhead. In this case, depending upon the system load, the amount of inter-cluster communication, and granularity of threads, the benefits of more parallelism (threads) may diminish.

The hierarchy is particularly effective in exploiting the communication locality that is concentrated within clusters. It is less effective when each processor communicates frequently with processors that are not necessarily a part of the same cluster.
The hierarchy has been used to implement multiprocessor systems. It is used in the KSR-1 [4] commercial system, and research prototypes, such as Hector [56] and NUMAchine [57]. The KSR-1 is a COMA shared-memory system, whereas NUMAchine is an implementation of a NUMA shared-memory system. The KSR-1 system is described in Section 2.2.1 and Section 2.2.2 contains a discussion on NUMAchine.

### 2.2.1 Kendall Square Research’s KSR-1

The KSR is the first commercial COMA architecture. The KSR architecture addresses the issue of scalability of implementing a logically shared address space across the physically distributed memories. The KSR approach to solve this problem is to treat the main memory as a cache, which is a COMA approach. Hence, the address space in the KSR architecture is not binding to any physical memory location. The address space of each process, called the Context Address (CA) space, is mapped to a System Virtual Address (SVA) space. The architecture provides $2^{64}$ bytes SVA. The target applications for the architecture include scientific as well as commercial database applications, such as transaction processing and decision support applications.

The KSR employs a hierarchy of rings to connect up to 1088 processing modules, called *cells* in KSR terminology. Each cell consists of a processor, a cache, a local memory, and a network interface. Since the SVA is not bound to any physical memory location, the local memory serves as a second level cache. The memory hierarchy comprises three levels of cache. The first two levels are local to each processor, and the third-level cache is accessed through the network. The caches that are local to each processor are: (i) a first-level cache called *data subcache*, and (ii) a second-level cache called *local cache*. The collection of the local caches of all remote processors form the third level cache that is accessible through the network.

The local cache is equivalent to the main memory, and the data subcache is equivalent to the processor cache of a non-COMA architecture. The 32-Mbyte local cache is organized as 128 sets of pages [29]. Each set is 16-way associative. A 16-Kbyte page is composed of 128 subpages, where the size of a subpage is 128 bytes. The subpage is the unit of transfer and cache coherence among local caches, and its size determines the coherence granularity. The 256-Kbyte data subcache is organized into 128 blocks of 2 Kbytes each. The blocks are stored in 64 2-way associative sets. A block is composed of 32 subblocks. The size of a subblock is 64 bytes.

The KSR-1 is an implementation of the KSR architecture. It uses a custom processor that runs at 20 MHz. The processor architecture is a RISC-like pipelined load-store architecture with separate integer and floating point functional units. The KSR-1 uses a hierarchy of rings to connect up to 1088 processors. A first level (local) ring connects up to 32 local caches, and
the second level (global) ring connects up to 34 local rings. The packet that moves in a slot contains a 16-byte header and 128 bytes of data (subpage). The bandwidth of the local ring is 8 million packets per second (1 GB per second). The global ring can be configured to operate at 1, 2, or 4 GB per second [28]. The network interface contains a directory of pages stored in the local cache. The directory has an entry for each local page, which contains coherence information about each of its 128 subpages. Each local ring is connected to a global ring through an inter-ring interface. The inter-ring interface contains a dual combined directory of all caches connected through its local ring. One copy of the combined directory is accessible from the local ring, and the other copy is accessible from the global ring. Since there is no permanent location for a subpage, the request packet needs to search the subpage in every node as it goes along a ring. The combined directories at inter-ring interfaces facilitate this search operation. When a request packet on a local ring does not find an entry in the combined directory of the inter-ring interface, it assumes that the requested subpage is located on a remote ring. The packet is then moved to the global ring. Similarly, when a request packet on the global ring finds an entry for a subpage in the combined directory of an inter-ring interface, it assumes that the requested subpage is located on the associated local ring. The packet is then switched to the local ring where it eventually finds the subpage.

The machine implements $2^{40}$ bytes SVA. The 40-bit process address, also called CA, is translated into a 40-bit SVA. The 40-bit SVA is used to locate the data in the system. First, the data subcache is searched for the data. In the case of a data subcache miss, the missing subblock is fetched from the local cache if the block (that holds the subblock) is already found in the data subcache. In the case of a missing block, the space for the whole block is allocated and the missing subblock is fetched from the local cache. The local cache and the subcache are always coherent. The local cache miss is handled in a similar manner. In the case of a read miss in the local cache, the missing subpage is fetched from a remote local cache. The address of the missing subpage is used to form a request packet that is circulated over the local ring. When a cell in the local ring finds a copy of the subpage in its local cache, it sends the page to the requester. When an inter-ring interface finds a request packet that contains an address which is missing in its combined directory, it forwards the packet to the global ring; otherwise it forwards the packet to the next cell. A packet traversing the global ring consults the copy of the combined directory, which is accessible from the global ring at each inter-ring interface to find out the local ring where a clean copy of the subpage is located. It then moves to the local ring for the target copy. When a subpage is moved to a cell, the directories at all concerned network interfaces and inter-ring interfaces are updated. This design incurs a huge increase in latency when a packet goes to the global ring. For example, in an experiment to study the global ring
contention, the worst latency to move a subpage within a local ring when all 32 nodes of the local ring are generating requests was found to be about 8.4 $\mu$s, whereas the latency to move a subpage across the global ring when only two nodes are active was found to be 25.2 $\mu$s [42]. Estimates of data rate per node under various load conditions show that the conflict free rate for a node is about 17 MB/s, which is reduced to 15 MB/s when all 32 nodes of the local ring are generating requests. When data moves between rings, rates are much lower. The range is 5 MB/s for one node without contention to about 4 MB/s when 32 nodes are active, and declining to less than 2.5 MB/s with a load of 60 nodes [42].

2.2.2 NUMAchine

The goal of NUMAchine architecture is to implement a multiprocessor system that is modular, cost-effective, scalable to a reasonable size, and easy to use efficiently [57]. The NUMAchine is a cache-coherent NUMA (Non-Uniform Memory Access) shared-memory multiprocessor with the memory distributed across the stations. Each station connects up to four processors, a memory module, an I/O interface, a network cache, and a station-to-ring interface through a shared bus. A flat physical addressing scheme is used with a specific address range assigned to each station. Thus, the address space is logically shared, but physically distributed across the stations. The time needed by a processor to access a given memory location depends upon the distance between the processor and the memory. Thus, the architecture is of a NUMA type. A network cache is used in each station to keep the cache lines that are fetched from remote memories by the local processors.

A hierarchy of bit-parallel slotted unidirectional rings is used to connect all the stations. In a two-level hierarchy, a local ring connects up to four stations, and a global ring connects up to four local rings. The system size can be increased by introducing more levels to the hierarchy. The use of a ring-based interconnection network has several advantages. For example, (i) there is a unique path between a pair of stations, (ii) the routing is very simple which allows fast operation of the ring, and (iii) the packet that traverses the whole ring is observable by all stations on the ring allowing a simple and natural multicast by means of a single packet. The NUMAchine architecture has fully exploited all these features of the ring in its communication and cache-coherence scheme.

Routing is very simple in NUMAchine. The routing mask in a packet is a bit vector which specifies the address of a destination node. The routing mask is divided into as many segments as there are levels in the hierarchy. The first segment, called the node segment, contains as many bits as the maximum number of nodes in any local ring. For all other segments the number of bits is the same as the number of rings at the corresponding level. Figure 2.4 shows
the routing mask for the hierarchy of Figure 2.3. The node segment contains 4 bits, one for each node in a local ring; and the ring segment contains 4 bits, one for each local ring connected to the global ring. To specify the address of a given node, the corresponding bits are set in the ring and node segments. For example, the address 0101 0100 specifies that the packet is to be sent to nodes (R2, N2) and (R0, N2). This scheme provides a simple mechanism for multicasting. Note that the routing mask may not specify multiple destinations exactly. For example, if 0101 0110 is used to multicast a packet to nodes (R2, N1) and (R0, N2), it will also cause nodes (R2, N2) and (R0, N1) to receive the packet. The multicast scheme is used to invalidate copies of a cache line to maintain cache coherence. The NUMAchine employs two levels of cache coherence protocol, one is the memory level and the other is the network cache level. The memory level protocol is described in Section 5.1, and the network cache level protocol is discussed in Section 6.1.

<table>
<thead>
<tr>
<th>bit 7</th>
<th>RING SEGMENT</th>
<th>bit 4</th>
<th>NODE SEGMENT</th>
<th>bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3</td>
<td>R2</td>
<td>R1</td>
<td>R0</td>
<td>N3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>N2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>N1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>N0</td>
</tr>
</tbody>
</table>

Figure 2.4: The routing mask

The communication protocol employed in NUMAchine is very simple[27]. The packet that traverses a ring is checked by every node and the inter-ring interface that encounters it. A node compares its own address with the destination address of the packet. If both addresses match, the node removes the packet from the ring and stores it in its receive buffer. If it is one of two or more destinations, it keeps a copy of the packet, resets the node and ring bits of the destination address corresponding to its own address, and then forwards the packet to the next node. Otherwise, the node simply forwards the packet if it is not the destination.

An inter-ring interface that connects a local ring to a global ring performs packet switching between the two rings. For a packet in the global ring, if the bit in the ring segment corresponding to its connected local ring is set, it copies the packet to the local ring. The packet is removed from the global ring by the inter-ring interface that has injected the packet into the global ring. There is a control bit, called up-bit, in the header, that indicates whether the packet should be switched to the global ring. When the inter-ring interface finds the up-bit of a packet set, it resets the up-bit and switches the packet to the global ring.

In NUMAchine, the possibility of a deadlock is prevented by distinguishing the message types and handling them differently. The messages are categorized on the basis of the need for a response. A message that elicits a response is called a non-sinkable message, and a message
that does not elicit a response is called a sinkable message. The examples of sinkable messages are all types of read response, write backs, and invalidations. All request messages are non-sinkable messages. The station-to-ring interface employs different queues to store the received sinkable and non-sinkable messages. By separating the messages, the sinkable messages are guaranteed a downward path to the target station, and are given priority over the non-sinkable messages.

Flow control is used to deal with packet losses or excessive packet delays caused by congestion in part or all of the network. The NUMAchine interconnect employs three levels of flow control [27]. At the top level, when the down-buffer in an inter-ring interface overflows, all inter-ring interfaces connected to the same global ring stop putting their packets into the global ring. A buffer is declared overflowing when it is filled to a certain level. The remaining space is left to store the incoming packets that are in transition. The size of the remaining space is calculated by taking into account the flow control signals' propagation delays. At the local ring level, two conditions are handled. First, when the up-buffer of the inter-ring interface overflows, all stations cease to transmit. Second, when the receive buffer of a station-to-ring interface overflows, all stations connected to the local ring stop transmitting and the inter-ring interface (connecting the local ring to the global ring) also stops discharging packets from its down-buffer. At the station level, when the receive buffer of a card on the station overflows, all other sources of packets originating within the station for that card are stalled.

The NUMAchine prototype is a 64-processor system using two levels of hierarchy. A station comprises four processor modules, an I/O board, and a network interface board. Each processor module consists of a 150 MHz MIPS R4400 processor, a 1-Mbyte secondary cache, and an external agent. The external agent is responsible for handling all data and commands in both directions: those that are generated by the processor and those that are sent to the processor. The network interface board consists of both the network cache and station-to-ring interface. Each station has a 4-Mbyte network cache, which is implemented using DRAM. The tag and state bits of the network cache are stored in SRAM. The memory module is implemented using DRAM that contains up to 2 Gbytes of memory. The memory is split into two banks and is interleaved. The directory information for cache coherence is maintained in SRAM. The station bus is 64 bits wide and runs at 50 MHz. A 50 MHz 64-bit wide unidirectional bit-parallel slotted local ring connects up to 4 stations. A global ring of the same capability connects the 4 local rings.
2.3 Bidirectional Rings

2.3.1 MetaRing

The MetaRing is a bidirectional ring-based local area network developed at IBM T.J. Watson lab [5]. The goal of this ring network is to integrate synchronous and asynchronous traffic and achieve high throughput by spatial reuse. The spatial reuse is the ring characteristic that allows concurrent transmission over distinct segments of the ring. The integration of synchronous and asynchronous traffic requires support for bandwidth reservation and multiple priority levels. The access control mechanism is a combination of slotted and buffer insertion ring. The ring starts as a slotted ring, but evolves into a buffer insertion ring. The buffer insertion ring allows variable packet sizes and achieves high utilization as all links can be kept at full utilization at all times, provided that the nodes have enough data to transmit. A drawback of the buffer insertion ring is starvation, when one or more nodes can wait to transmit a packet for an indefinite period of time. This may happen when a node is completely shut out by some upstream nodes that always send packets to some of its downstream nodes. The starvation can be avoided by ensuring fairness in bandwidth allocation to every node. The MetaRing employs a global fairness algorithm. The algorithm is distributed and tolerates faults caused by a missing control signal because every node is capable of regenerating the control signal. It uses a control signal, SAT, that circulates in the direction opposite to the data packets it controls. Two parameters, \( k \) and \( l \), control the movement of the SAT signal through the nodes. Every node maintains the count of packets it has transmitted since releasing the last SAT signal. A node is allowed to transmit at most \( k \) packets between transmitting two successive SAT signals. After transmitting \( k \) packets, it stops and waits for either the arrival of the SAT signal or time-out. If time-out expires, it regenerates the SAT signal, resets the packet count, and forwards the signal to the next node. The node can hold the SAT signal, which it receives from its downstream neighbor, if its transmit buffer is not empty and the packet count shows that it has sent less than \( l \) packets since the last transmission of the SAT signal; otherwise it forwards the SAT signal and resets its packet count. When a node holds the SAT signal, its transmission quota is reduced from \( k \) to \( l \) packets. This algorithm can be extended to incorporate the support for multiple priority levels. The proposed network has full-duplex serial links. The prototype supports the transmission of variable size packets at 100 MB/s link speed and with aggregate throughput of 700 Mb/s [5].
2.3.2 GigaRing

The GigaRing channel is a bidirectional ring that connects nodes through high-speed point-to-point links [44]. The bidirectional ring is used for its resiliency to failures and support for hot-swapping. The channel is designed for the transfer of large blocks of data. Cray computers are using this channel as a scalable interconnect for I/O subsystems, which is expected to replace the expensive switches of the existing I/O subsystems. The GigaRing supports three main functions: direct memory access (DMA) transfers, messaging, and remote node maintenance and configuration. In a DMA transfer, a server node is allowed to directly read or write a block of memory in a remote client node. The messaging is a normal client initiated request-response transaction. Remote maintenance supports node maintenance and configuration remotely through memory-mapped register interfaces.

The channel is implemented using a pair of unidirectional rings running in opposite directions. The network interface is an ASIC chip that implements the physical and logical layers of an SCI-based channel protocol. A pair of 32-bit input-output links provide the physical interface to the ring. The chip contains two transmission and two receive buffers, one for each ring. Both transmission buffers are connected to a set of request and response buffers. The packet to be transmitted is fetched from the node across the port and stored in either a request or response buffer depending upon the type of packet. The transmission rate is 150 Mbps per signal, providing 9.6 Gbps aggregate raw bandwidth to every node.

As in SCI, the insertion ring access mechanism is used in the GigaRing channel, which allows transmission of packets of different sizes. The unit of transfer is a packet that contains a 16-byte header and an optional payload of up to 256 bytes. A packet is decomposed into units of 32-bit symbols, which are transmitted across the channel. The channel protocol requires a node to acknowledge the source when it receives the packet. The source node retains the packet in its transmission buffer for a possible retransmission in the case of negative acknowledgement. The number of concurrent transmissions of packets depends upon the number of transmission buffers in the chip. The node starvation is prevented by employing a bandwidth allocation scheme that ensures fairness. A node requires a go-bit for transmission. When a node is ready for transmission it holds the go-bit that it receives from the downstream node, and waits for the insertion buffer to be empty. The upstream nodes will eventually stop their transmission and give the node a chance to transmit packets. An SCI-based scheme is employed to avoid deadlock by using separate request and response buffers. The chip contains two request and two response buffers, one each for read and write. The congestion control mechanism of SCI does not provide congestion control at end-points, which is important for DMA transfer. The GigaRing channel
protocol controls the end-point congestion by limiting the number of outstanding requests (the requests awaiting response) at each node.
In this thesis, the bidirectional ring is proposed as an interconnection network for a shared-memory multiprocessor system. The attractive features of the bidirectional ring are its ability to exploit communication locality found in a large number of parallel programs to enhance their performance, and its resiliency to failure. A bidirectional ring consists of two rings transmitting packets in opposite directions, as shown in Figure 3.1. The constituent rings are bit-parallel and use the slotted access protocol. A source node selects the ring to transmit a packet based on the shortest distance to the destination node, which is measured as the number of hops between them. Thus, the length of the transmission path between any two nodes is at most half the ring size. In this transmission protocol, request and response packets of a transaction involving two nodes traverse different rings.

Figure 3.1: An example of the bidirectional ring

The bidirectional ring has the ability of adapting to the communication pattern of the application program by forming clusters of processors dynamically. If multiple threads of a
process are scheduled to run on a set of adjacent processors, these processors automatically form a cluster where all request-response transactions among the processors involve communication within the segment of the ring connecting these processors. A cluster forms as a result of a logical grouping of processors, and is called a *dynamic cluster*. Hence, the number of processors forming a cluster is not constrained by the physical structure of the network. The only condition for forming a dynamic cluster is that the processors allocated to the set of threads be adjacent. Consider three processes that are allocated processors as shown in Figure 3.1. Each of the four sets of adjacent processors forms a dynamic cluster. The processor assignment for processes 1 and 3 engenders one dynamic cluster for each of these processes. Thus, the communication among the threads of each process tends to be localized to its own cluster, but some traffic, such as coherence traffic, may span the whole ring. In contrast, the processor assignment to process 2 forms two discrete clusters. The communication among the threads of process 2 not only causes intra-cluster traffic, but also generates inter-cluster traffic. The inter-cluster traffic interferes with the traffic of only those clusters that are in the transmission path. In a multiprogramming environment these processes could belong to different application programs.

The notion of hierarchy is very powerful in supporting a NUMA architecture. The dynamic clustering allows the bidirectional ring to uphold the abstraction of hierarchy. It offers two advantages over the static clustering of the hierarchy of unidirectional rings. First, the bottleneck of the hierarchy is removed, because there is no upper-level ring to connect the clusters. Second, there is no spanning overhead, which is likely to improve the performance of a process that would be allocated processors across two or more local rings if run on a hierarchy of unidirectional rings. In the hierarchy, a process that needs as many processors as there exist in a local ring can naturally fit in one local ring, but the need for one more processor will cause the spanning of two local rings. This introduces a communication overhead proportional to the size of the second-level ring connecting the two local rings. We call this the *spanning overhead*. If a packet must cross more levels of hierarchy to reach the destination ring, then each level will add an overhead proportional to the size of the ring at that level. Furthermore, when a packet ascends or descends the hierarchy it passes through several inter-ring interface buffers, each of which contributes a variable amount of *buffer delay* to the overhead. The buffer delay depends upon many factors, notably the communication pattern and the network utilization. The buffer may also lose packets if it overflows, which can be prevented by controlling the flow in the network. In the case of high network utilization, a flow control mechanism may be triggered to prevent packet losses, which effectively increases the overhead. Thus, the spanning overhead is generally more than the propagation delays due to additional links, which causes a significant increase in the communication latency. In contrast, expanding the cluster in the bidirectional
ring to adjust to the number of processors needed will only increase the communication cost proportional to the propagation delays due to additional links. Although the latency grows logarithmically due to additional links in the hierarchy, as compared to the linear growth in the bidirectional ring, the spanning overhead overshadows the advantage of slow growth rate.

A dynamic cluster in a bidirectional ring is a chain-like structure of nodes. The round trip delay within a cluster varies depending upon the positions of the source and the destination nodes. The maximum round trip delay is \(2(P - 1)\), where \(P\) is the number of nodes in the cluster. In contrast, a local ring with the same number of nodes will induce a round trip delay equal to \((P - 1)\). This difference in the round trip delay is non-negligible as long as the size of the cluster is equal to the size of the local ring. For larger clusters that span multiple local rings, the spanning overhead will overshadow this difference in round trip delay.

Reliability is an important concern for many parallel applications. The bidirectional ring is inherently more reliable, because the dual rings allow it to survive a ring failure. It can also tolerate a node failure by diverting traffic from one ring to another at either side of the failed node. The reliability issue and its impact on cache coherence protocol is fully discussed in Chapter 7.

3.1 Routing

The routing scheme designed for the hierarchy of unidirectional rings can also be used effectively for the bidirectional ring. To implement this scheme, a hierarchy is logically mapped onto the bidirectional ring. An equivalence of a local ring can be formed by grouping a number of adjacent nodes, called a local routing cluster \((Rc)\). A group of local clusters can form a second-level cluster, and so on, to extend the level of the hierarchy. Thus, each routing cluster may be likened to a ring of the corresponding level in the hierarchy. For example, Figure 3.2 shows four local routing clusters that are created by logically mapping the hierarchy of Figure 2.3. Each routing cluster contains four nodes. Clusters and nodes within the clusters are assigned numbers in ascending order following the counter-clockwise direction. A destination node address 0010 0001 specifies that \((Rc1, N0)\) should receive the packet. The routing tag for multicasting a packet to nodes \((Rc2, N1)\) and \((Rc0, N2)\) is 0101 0110. The propagation of this tag will cause nodes \((Rc2, N2)\) and \((Rc0, N1)\) to receive the packet too. Hence, as in the case of hierarchy, the multicasting on the bidirectional ring is not precise.

Routing clusters are fundamentally different from the dynamic clusters. The dynamic cluster is just a grouping of processors characterized by the communication pattern of a particular application program. In contrast, routing clusters are formed at design time to provide the
addressing needed to route packets through the ring. Hence, their size and configuration are fixed.

Figure 3.2: Routing scheme for the bidirectional ring

The ring to be used for the transmission of packets can easily be determined from the routing tags of the source and destination addresses. There are three cases to be considered: First, when both nodes are within the same cluster then all the nodes denoted by bits to the left of the source node bit are accessible through the counter-clockwise ring, and those to the right through the clockwise ring. Second, if the two nodes are in different clusters, then some clusters are obviously accessible through one ring or the other; for instance all nodes of Rc1 are accessible through the counter-clockwise ring from all nodes of Rc0. Third, there is one cluster corresponding to every cluster that is diametrically opposite to that cluster. In this case, depending upon the position of the source node in the cluster, some nodes of the opposite cluster are accessible through one ring, and the rest through the other ring. For instance, if the source node is (Rc0, N1) then nodes (Rc2, N0) and (Rc2, N1) are accessible through the counter-clockwise ring, while the nodes N2 and N3 of Rc2 are accessible through the clockwise ring. A simple combinational circuit can be designed to determine the ring from the relative positions of the routing bits of the two nodes.

A simple network interface can be designed for the bidirectional ring. The number of external connections needed to maintain the bandwidth of a local ring in each direction is almost double the connections required to interface with a local ring in the hierarchy. A sample interface is described in Chapter 7 that incorporates features to support reliable operation of the bidirectional ring.
We have simulated the bidirectional ring and the hierarchy of unidirectional rings of different sizes to assess their relative performance using a simple but real communication protocol and synthetic workloads that are tuned by carefully selecting the parameters to reflect the actual workloads. A similar approach has also been used in evaluating the performance of a hierarchical ring-based system [17]. The use of synthetic workloads allows us to study the behavior of a network in many interesting situations, and it also makes the simulation of large-scale networks manageable. This preliminary evaluation of the bidirectional ring in comparison with the hierarchy helps us to develop a better understanding of the bidirectional ring, and it is also beneficial in exposing the potential problems that can occur in implementing large systems running certain types of applications. Scott et al used a different approach by first developing an analytical model of the SCI ring and then using a simulation of the actual network driven by a simple synthetic workload to analyze and validate the analytical model [43]. Section 4.1 describes the system model. The network protocol is discussed in Section 4.2. Section 4.3 explains different workload models used in the simulation. The performance results are discussed in Section 4.4.

4.1 System Model

A node consists of a processor, a memory unit and a communication switch. The network is connected to the memory through the switch, and receives only those memory requests that are not satisfied by the local memory. This is modeled by using a fraction of all requests generated by the processor as requests to remote memories. The request rate is a parameter that indicates the average number of requests generated by the processor per cycle. It is fixed at 0.05, which corresponds to 20 cycles between two consecutive cache misses. This choice is supported by a recent study of a number of application programs that observed a mean number of processor cycles of between 6 and 137 for shared data reads [3]. We assume that code and private data
references (such as to a stack) always hit in the local cache. Factoring in shared data writes and shared data cache hits yields a more realistic mean number of processor cycles between cache misses of at least 20. The ring cycle time and the processor clock cycle time are considered to be the same. One memory access is assumed to take 10 processor cycles. Thus, reading a word from the memory takes 10 processor cycles. For a cache line read, the first word takes 10 cycles, but each subsequent word is available after every 5 cycles. The ring is a slotted ring [18].

The communication locality can be modeled as groups of nearby nodes around a source processor and assigning an access probability to each group indicating that the target memory is in that group when the request is not to the local memory. This model is similar to the clusters of communication locality model [17]. The communication locality defines the local memory target probability, the number of groups, the size of each group, and the access probability of each group. A group of size $s$ and access probability $p$ consists of all nodes within a distance of $s/2$ from the source. A request generated at the source goes to a node in this group with a probability $p$ given that the request is not to the local memory. The distribution of destination addresses within a group is uniform. For instance, in a system of 1024 processors with a local memory target probability of 0.9, $GS = (512, 768, 1024)$ and $GP = (0.8, 0.95, 1.0)$ defines 3 groups, where $GS$ and $GP$ indicate group sizes and their corresponding access probabilities. The local memory serves on average 90% of all requests. The first group consists of 512 nodes and 80% of the remaining remote requests involve nodes within this group. The second group is formed by adding 256 more nodes to the 512 nodes of the first group, thus having 768 nodes. The access probability of this enlarged group is 0.95, meaning that 95% of remote requests go to the nodes of this group. Finally, all 1024 nodes form the third group, and all remote requests are within this group.

The invalidation packets introduced by the cache coherence protocol are not considered in the network traffic. Ignoring the invalidation traffic has no major effect on the overall results. For common applications, the average number of shared writes causing invalidations is not more than 2 to 3 percent of the total requests, and the average invalidation per shared write involves not more than 2 packets [13]. Assuming one packet per invalidation, the percentage of invalidation packets becomes less than 10% of the total request packets. This value decreases further if it is considered as a fraction of the sum of request and response packets.

### 4.1.1 Simulation Methodology

We wrote a simulator using the smpl simulation library [34], that simulates the system behavior on a cycle-by-cycle basis. The batch means method of output analysis was used to record the statistics. In batch means method a single long run is divided into batches. We have discarded
the first batch to account for initialization bias. A separate sample mean was computed for each batch. These batch means were then used to compute the grand mean and confidence interval [34]. We used a 95% confidence interval for the grand mean. The batch termination criterion requires every processor to complete some minimum number of requests.

### 4.2 Request-Response Protocol

A Hector-like request-response protocol [56] is used for network transactions. A subset of possible transactions is used that is sufficient to give a good indication of the relative performance that may be expected from the bidirectional ring. The three types of requests modeled are: i) READ a memory word, ii) READ a CACHE LINE (Cache line transfer), and iii) WRITE a memory word. Each processor can have only one outstanding request at a time. We assume that 80% of all requests are reads, 20% are writes, and 70% of reads cause a cache line transfer. For the read cache miss probability we assume 0.7 throughout the study, which is consistent with empirical statistics [10]. A single packet is used to transmit a request. The response can involve multiple packets, such as for a READ CACHE LINE. The data to be written is included in the WRITE packet. For writes, the acknowledgements are issued after queuing the WRITE packets for later storing in the memory. A negative acknowledgement (NACK) is issued when the request cannot be buffered at the receiving node. The sender retransmits the same request after receiving the NACK. Another occasion when a sender retries is after a time-out, which is a mechanism used to recover from a packet loss. After a time-out expires the sender assumes that the packet is lost. It then resends the request packet. The time-out period should be large enough to avoid unnecessary retries, but should not be too large to increase the transaction latency. We have selected the time-out period after several iterations, which neither generates unnecessary retries, nor significantly increase the latency. There is only a one packet transmit buffer. When a node is ready to transmit a packet but does not find an empty slot in the ring, it is said to be **blocked**, and the time it waits for an empty slot is called the ring **access delay**. During this period the node continues to serve requests from remote processors.

In the hierarchy of unidirectional rings a WRITE packet is turned into an acknowledgement packet by simply changing the request type. In a bidirectional ring this scheme cannot work without losing the advantage of shorter distance for the response. A separate acknowledgement packet is formed and transmitted on the ring running in the opposite direction. This requires an acknowledgement buffer for each ring at every node to queue the acknowledgement packets for later transmission when the network is busy. We used this simple protocol to highlight the potential problem spots in the network structures. We used 256-deep receiving, acknowledgement,
4.3 Workload Models

A number of workloads have been used to model different request traffic patterns. Holliday and Stumm show that substantial ring contention is observed when more than 10% of memory requests are to remote memory modules [17]. Hence, to examine the network’s effect on latency, 80% of requests are targeted to the local memory, and 20% are transmitted through the network to remote memories. The salient features of the workload models are discussed below.

1. **Bursty:** This workload models multiple request packets transmitted in a burst, such as for cache line writes, multiple outstanding requests, and cache line prefetching. The two parameters that characterize this model are the average number of packets in a burst (burst length), and the average interval between two bursts. The average burst length is set to 5, and the average interval between two bursts is fixed at 100. All the packets in a burst are directed to the same destination. This is because: i) a cache line write is sent to a single destination, and ii) due to spatial locality consecutive cache misses are served by the same target memory.

2. **Uniform:** This workload model selects the destination node randomly within a group.

3. **Hotspot:** There are periods when requests originating at different processors are destined to the same memory module due to synchronization or false sharing. In such a situation a single node receives a large number of requests from many nodes. Different hotspots are selected randomly, each for 10% of the total hotspot transactions. Hotspot transactions are 3% of the total requests, which is in the range of practical values [9]. The other 97% of packets are generated using the uniform workload model.

4. **Mirror:** The destination address is at the same distance from the other extreme as the source is from one extreme, that is \( \text{destination} = (N - \text{source}) \mod N \), where \( N \) is the number of nodes. This model does not exhibit communication locality, thus it serves as the stress test for a network designed to exploit communication locality. For this workload 2% request rate is assumed because a large amount of traffic flows across the whole network, the network saturates at higher request rates.
4.4 Comparing Performance of the Bidirectional Ring and the Hierarchy of Unidirectional Rings

The main performance metric used to compare the performance of different networks is the request latency, which is defined as the number of cycles spent from the time when a processor issues a request to the memory to the time a response is received from the memory. Thus, the request latency is a sum of blocking times at both ends (source and destination), network propagation delays, and cumulative time spent in all the inter-ring interface buffers by the request and the response packets. Different memory requests experience different latencies. The average latency for each request type is computed first, and then the weighted averages of the three types of latencies are calculated.

Another important performance measure is network utilization, which is defined as the average percentage of busy slots over the total number of slots in the ring at any given time. This average is calculated over the total number of execution cycles. For the results presented in this section, a single application is assumed to be running.

Systems of various sizes, ranging from 4 to 1024 processors, have been simulated using the above workload models. Following are the three different network configurations considered.

1. **Hierarchy** is a hierarchy of unidirectional rings where 4 processors are attached to each of the lowest-level rings. Table 4.1 shows the configuration of different sizes in terms of the branching factor of each level of the hierarchy, from the lowest to the highest level. We call the lowest-level rings local rings, and the highest-level ring a global ring. This configuration gives a balanced hierarchy.

<table>
<thead>
<tr>
<th>No. of Processors</th>
<th>Hierarchy</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>4,2</td>
</tr>
<tr>
<td>16</td>
<td>4,4</td>
</tr>
<tr>
<td>32</td>
<td>4,4,2</td>
</tr>
<tr>
<td>64</td>
<td>4,4,4</td>
</tr>
<tr>
<td>128</td>
<td>4,4,4,2</td>
</tr>
<tr>
<td>256</td>
<td>4,4,4,4</td>
</tr>
<tr>
<td>512</td>
<td>4,4,4,4,2</td>
</tr>
<tr>
<td>1024</td>
<td>4,4,4,4,4</td>
</tr>
</tbody>
</table>

2. **Bi-1** is a bidirectional ring where the channel width of each of the two rings is half the
channel width of a unidirectional ring in the hierarchy. This is simulated by doubling the number of packets per cache line transfer on the bidirectional ring in comparison with the hierarchy. A request is assumed to be transmitted in a single packet.

3. Bi-2 is a bidirectional ring where each link has the same width as the ring in the hierarchy.

Two models of communication locality are considered, where the system consisting of $N$ nodes is divided into three groups of communication locality:

1. CLM1: Local memory target probability=0.8, $GS = \{N/2, 3N/4, N\}$, and access probability is assumed to be $GP = \{0.8, 0.95, 1.0\}$. This represents an application showing low communication locality.

2. CLM2: Local memory target probability=0.8, $GS = \{4, 20, N\}$, and the access probability is assumed to be $GP = \{0.8, 0.95, 1.0\}$. This is representative of applications showing very high communication locality.

4.4.1 Network Performance

In this section, the simulation results for all workload models—uniform, bursty, mirror, and hotspot—are presented. For brevity, the term hierarchy is used for the hierarchy of unidirectional rings, and latency for the average latency.

4.4.2 Uniform Workload

The performance of the hierarchy and Bi-1 is not significantly different for small networks under uniform workload, as shown in Figure 4.1. Latency in the hierarchy becomes considerably worse for large networks, mainly because of congestion at higher-level rings. In this situation inter-ring interfaces become points of high activity, as shown in Figure 4.2. It is interesting to observe the effects of high utilization of the high-level rings in the hierarchy in comparison with the bidirectional ring. In the hierarchy, the high utilization of high-level rings causes inter-ring interfaces to drop packets. The requester observes the packet drop after the time-out period expires, even though the packet that has been dropped is a response packet. The time-out period is set large enough to cover the round-trip time of a request-response transaction and also to avoid unnecessary time-outs due to long packet delays inside the network. When more nodes experience packet losses, the load on local rings decreases, as is evident by very low utilization of the local rings in Figure 4.2. In contrast, high utilization of the constituent rings in a bidirectional ring shows high contention for the network, but does not result in loss of packets because there is no switch inside the network, and consequently no buffer where packets
may be dropped. We observe that the global ring of the hierarchy is running at the saturation point; because it connects a large number of small rings in a hierarchical fashion.

Figure 4.2 shows a marginal difference between the utilization of the two rings in the bidirectional ring, which is due to the fact that different rings are used for transfer of request and response packets between any two nodes. Hence, the network load tends to be balanced between the two rings.

The causes of packet loss in the two networks are different. In the hierarchy, packets are lost mainly due to the lack of buffer space at the inter-ring interfaces. In the bidirectional ring packets are lost when acknowledgement packets are dropped due to the shortage of acknowledgement buffer space. This occurs very seldom, and appears only in those cases where momentary bursts of requests are received by some nodes. Therefore, packet loss is only observed in the hotspot workload.

Bi-2 results in improved performance for networks of all sizes, as shown in Figure 4.1.

![Graph showing latency for uniform traffic and CLM1](image)

Figure 4.1: Latency for uniform traffic and CLM1

The hierarchy is very sensitive to communication locality. In CLM2, 96% of communication takes place within a distance of 4; that is, most of the communication is confined to the local ring. Comparing graphs of Figures 4.1 and 4.3, it is obvious that the hierarchy shows lower latency for CLM2 than CLM1. The high communication locality causes low traffic in a high-level ring and consequently no congestion at inter-ring interface buffers, hence the spanning
overhead is low and the growth of latency is logarithmic in the hierarchy. The response of the bidirectional ring is also favorable to high communication locality, and its performance is comparable to the performance of the hierarchy for up to 256-processor system. For large systems, it performs worse than the hierarchy, because the effect of linear growth in its latency becomes more pronounced as opposed to the logarithmic growth for the hierarchy. The effect of locality is also reflected in the ring utilization graph of Figure 4.2. In particular, the utilization of high-level rings of the hierarchy is significantly reduced from the case of CLM1, which is a consequence of concentration of communication within a short distance of 4.

4.4.3 Bursty and Mirror Workloads.

Results for the bursty workload are shown in Figure 4.4. The latency of the hierarchy for small systems is slightly less than the latency of Bi-1. For large systems the hierarchy shows much worse latency.

Lack of communication locality in the mirror workload increases the latency in both types of networks, as shown in Figure 4.5. The adverse effect is more pronounced in the hierarchy, because a large number of transactions occur between the nodes located in different rings which causes packets to traverse several levels. Every time a packet moves either up or down in the
Figure 4.3: Latency for uniform traffic and CLM2

Figure 4.4: Latency for bursty traffic and CLM1
hierarchy, it may spend some time in an inter-ring interface buffer.

![Figure 4.5: Latency for mirror traffic and CLM1](image)

### 4.4.4 Hotspot Workload

The relative performance of the bidirectional rings and the hierarchy for the hotspot workload shows a slightly different trend than found for other workload models, as indicated in Figure 4.6. Unlike other workloads, the latency of Bi-1 is slightly higher than the latency of the hierarchy. Bi-2 still shows lower latency than the hierarchy. The main reason for the lower latency of the hierarchy is the large number of retries that appear on the bidirectional ring. In the hierarchy, a request packet passes through several buffers, one at each inter-ring interface, before reaching the destination node. This spreads out the load at the buffer of the destination node. In contrast, a single buffer in the bidirectional ring receives an equivalent number of requests. This results in many requests being dropped after reaching the destination, which in the hierarchy may find space in one of the inter-ring interface buffers. Another important difference is the contribution of retries to the latencies on the two networks. In the hierarchy a request may be dropped by an inter-ring interface, which adds less time to the latency than would be the case if the request is dropped by the destination node. In contrast, a request may be dropped only after reaching the destination in the bidirectional ring, and thus every retry adds a round trip delay to the latency.
Summary

The results of this chapter provide a preliminary evaluation of the performance of the bidirectional ring in comparison with the hierarchy. We used a simple, but practical, model of the system and synthetic workloads to drive the simulation, which enabled us to simulate large systems of up to 1024 processors. The major results of this evaluation can be summarized as follows:

1. The performance of the hierarchy is sensitive to communication locality. For high locality, where the communication is mostly concentrated within a short distance of the source processor, the latency of the hierarchy is significantly reduced. The bidirectional ring also exploits communication locality, and it shows better performance than the hierarchy for low communication locality for up to 256-processor system.

2. The buffers at inter-ring interfaces may cause delays and also a loss of packets when no flow control mechanism is used, as is the case in our simulations.

3. The high-level rings show high utilization than the low-level rings unless communication locality is high, which concentrates the communication within a short distance of the source processor. For the bidirectional ring, the utilizations of the constituent rings tend
to be close to each other.

4. The receive buffer in the bidirectional ring may lose packets for hotspot workloads if no flow control mechanism is employed. In contrast, the inter-ring interface buffers distribute the hotspot load in the hierarchy.
In this chapter, first the essential features of a cache coherence protocol for a hierarchy of unidirectional rings are discussed in Section 5.1. Then, a similar protocol for the bidirectional ring is described in Section 5.2. Finally, an implementation scheme of the cache coherence protocol on the bidirectional ring is described in Section 5.3. A detailed description of the cache coherence protocol for a hierarchy can be found elsewhere [11, 8].

Both protocols use a standard invalidation-based write-back scheme [49], where a processor must obtain an exclusive right before modifying a line in its cache. A cache line can be in one of three states: an invalid state meaning that the line is not valid, a shared state meaning that the line is read only, and an exclusive state meaning that the processor can write into the line. There is a locked state corresponding to each shared and exclusive state indicating that the cache is waiting for the data to arrive in response to either a shared-read or an exclusive-read request. The memory block which keeps a permanent copy of the line is called the home location of that line, and the memory module is called the home memory. The home memory maintains two states for every memory block. When the home location holds a valid copy of the line, then the line is in the shared state, otherwise it is in the dirty state, meaning that the line is dirty in some cache. The home memory also keeps a directory that has an entry corresponding to every line. A directory entry contains a routing mask denoting all the caches having a copy of the line. All requests for the exclusive right to a line are moderated by the home location. The coherence action involves invalidating all copies of the line in other caches before granting the processor the exclusive right. When a request is sent to the home location from a processor's secondary cache, then the corresponding location for the line in the secondary cache is locked until the response is received. The line is written back to its home location when it is replaced by another line in the cache. Table 5.1 gives some important request/response transactions.
5.1 Cache Coherence Protocol for the Hierarchy

The coherence protocol for the hierarchy exploits the natural ability of the ring hierarchy to support multicasting of packets that constitute the invalidation commands. In response to an exclusive-read request, the corresponding cache line must be copied into the requesting processor's cache either from the home location or from another cache that may have the dirty copy of the line. If the home location holds a valid copy, then it sends the data to the requester. The requester keeps the received line locked until it receives the invalidation packet which it interprets as the acknowledgement. The home memory multicasts an invalidation packet to all caches that may have a copy of this line as well as to the requester. The invalidation request packet is sent to the highest level in the hierarchy from which the invalidation packet can reach all of the desired caches. From this level, say the global ring in Figure 2.3, the invalidation packet is distributed to the lower levels as follows. As the invalidation packet visits each inter-ring interface while traversing the global ring, the interface transmits a copy of the packet to the connected local ring if the bit corresponding to the local ring in the routing mask of the packet is set to 1. Each ring is traversed fully by such a packet; the packet is removed from the ring by the interface at which it entered that ring. When the invalidation packet returns to the originating home memory, it is interpreted as an acknowledgement that the invalidation operation has been completed. If the home location of the line is locked (to prevent further accesses) at the time the invalidation request is issued, and if it is unlocked upon the return of the invalidation packet, it can be shown that the protocol will implement sequential consistency [8]. The above describes just one example of the invalidation process; it is intended to give the reader an indication of a typical transaction involving invalidation.

A memory consistency model gives the programmer a consistent view of the memory. The sequential consistency model imposes a system-wide ordering on all shared accesses. Three features of the hierarchical ring structure are important in maintaining the necessary ordering among all accesses. These features are: (1) there is a unique path between any two nodes in the hierarchy; (2) it is impossible for two packets to overtake each other in the network; and (3) packets are processed at each node in the order in which they arrive. The strict ordering on invalidations can be achieved by causing each invalidation request to spawn an invalidation packet only when the request (generated by the home memory module) arrives at the inter-ring interface of any ring that invalidation packets must traverse during the required multicast operation. Thus, the inter-ring interface acts as a sequencer. The presence of a sequencer and the above mentioned locking scheme ensures sequential consistency for the above protocol [8].
Table 5.1: Some important request/response transactions

<table>
<thead>
<tr>
<th>Request/Response</th>
<th>Function performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared-read</td>
<td>The copy that is read is sharable; as a result multiple copies of the cache line may exist.</td>
</tr>
<tr>
<td>Exclusive-read</td>
<td>All copies are invalidated before the cache line is read; as a result only a single copy exists.</td>
</tr>
<tr>
<td>Upgrade</td>
<td>An exclusive right to a line which was previously read in the shared state is obtained. Other copies, if they exist, are invalidated before granting the exclusive right.</td>
</tr>
<tr>
<td>Shared-intervention</td>
<td>The line is dirty in some other cache, which transfers a copy to the home location and the requester, and changes its own copy to the shared state.</td>
</tr>
<tr>
<td>Exclusive-intervention</td>
<td>The line is dirty in some other cache, which transfers the line to the requester and invalidates its own copy; as a result a single copy of the line exists with the requester.</td>
</tr>
<tr>
<td>Invalidate</td>
<td>Invalidation request is sent to all the caches to invalidate their copies of the line, issued by the home location.</td>
</tr>
</tbody>
</table>

5.2 Coherence Protocol for the Bidirectional Ring

Now, we will present a cache coherence protocol for the bidirectional ring. The protocol is based on the cache coherence protocol suggested for multiprocessors that use a hierarchy of unidirectional rings [8]. It follows the general framework of the invalidation-based write-back scheme. However, three major differences of the bidirectional ring impact the protocol design. First, there is no node in the bidirectional ring that can be used as a sequencer. Second, the existence of two paths between any two nodes in the bidirectional ring does not guarantee the strict sequencing of response messages. For example, an invalidation message can outrun a response message for a prior shared-read request. Third, the support for multicasting in the bidirectional ring is not straightforward. A scheme to multicast invalidation packets is described in Section 5.3. This scheme is based on the routing facility for multicasting a packet as described in Section 3.1.

The proposed protocol differs from the coherence protocol for the hierarchy in two ways. First, the invalidations are handled differently. Second, an invalidation pending state is introduced to sequentialize an invalidation message that outruns the response for a prior shared-read request. The steps for processing an exclusive-read request and an upgrade request, the two transactions that potentially involve invalidation, are elaborated in Figures 5.1 and 5.2, respectively. The shared-read request is handled in a way similar to the hierarchy with a minor modification to accommodate the invalidation pending state as described below. All other
transactions are performed in the same way as in the hierarchy.

To illustrate the invalidation process, consider an exclusive-read transaction such that the home location holds a valid copy of the line. Upon receiving the exclusive-read request, the memory locks the home location and sends the data to the requester. After receiving the data, the requester waits for an acknowledgement from the home memory to proceed. After sending the data, the home memory multicasts an invalidation request over the ring that is used to send the response to the requester, to selectively invalidate all copies of the line. When the invalidation packet returns to the home memory, the memory assumes that all copies have been invalidated and it unlocks the home location. Then, the home memory sends an acknowledgement to the requester to proceed, thus granting it the exclusive right to the line.

In the above scheme, an invalidation may outrun the response to a prior shared-read request. To illustrate this consider two processors, A and B, sending a shared-read request and an upgrade request, respectively, for the same cache line to the home memory, such that they receive the response packets from the memory over two different rings. These requests are processed by the home memory in order of their arrival. Since there are two different paths for the response packets to arrive at A, it may receive the invalidation before receiving the cache line in response to its prior shared-read request. Obviously, A cannot be allowed to disregard the invalidation; otherwise after the completion of the line transfer to A incoherent copies may exist at A and B. Processor A can use the received data once without violating the sequential consistency, because its shared-read preceded the upgrade request that caused the invalidation. A shared-intervention can face the same problem. After sending the line to the requesting processor, the processor that had the dirty copy may issue an upgrade request for the line which is now in the shared state. The upgrade request may cause an invalidation to be sent on the other ring from the one that is used by this processor to send the line to the requesting processor. The invalidation can arrive at the requesting processor before it receives the line. This may cause the same problem as described above for the shared-read transaction. While the same problem arises in two different transactions, from the requesting processor’s perspective both events are caused by a shared-read request. This means that it suffices to introduce an additional state to delay the invalidation when the cache is waiting for a shared-read response.

The invalidation can be delayed by introducing an invalidation pending state, which indicates that an invalidation is pending for the line that is to be received in response to a prior shared-read request. As shown in Figure 5.3, if a cache receives an invalidation request for the line which is in the shared locked state, then it changes the state of the line to the invalidation pending state. Upon receiving the line, it services the processor’s read miss and then invalidates the line. A complete algorithm for the shared read transaction is given in Figure 5.4.
Exclusive-Read Request
begin
1. The requester sends an exclusive-read request to the home location, and waits for the response.
2. Upon receiving the exclusive-read request the home memory locks the memory block.
3. if the block is in the shared state
   then
   (i) The home memory sends the cache line to the requester in multiple packets. The requester assembles the cache line and updates the cache, but keeps the line locked.
   (ii) The home memory circulates the invalidation packet over the same ring which is used for sending the cache line to the requester, to invalidate all the caches having a copy of the corresponding line. The invalidation packet returns to the home memory after traversing the ring.
   (iii) The home memory sends the invalidation acknowledgement to the requester, updates the directory, and unlocks the memory block.
   (iv) The requester unlocks the cache line after receiving the invalidation acknowledgement packet.
elseif the block is in the dirty state
   then
   (i) The home memory sends an exclusive-intervention request to the cache holding the dirty copy, call it dirty cache.
   (ii) The dirty cache sends the cache line to the requester in multiple packets as the intervention response, then sends an acknowledgement to the home location to proceed, and finally invalidates its own copy of the line.
   (iii) The requester stores the line in its own cache, which is assembled from the intervention response packets.
   (iv) After receiving the acknowledgement from the dirty cache, the home location unlocks the memory block and updates the directory.
end

Figure 5.1: Algorithm for processing exclusive-read request
Upgrade Request

begin
1. The requester sends an upgrade request to the home location, locks the line in its cache, and waits for the response.
2. After receiving the upgrade request the home location locks the memory block and then circulates the invalidation packet on the same ring which is used for sending the response to the requester.
3. The invalidation packet invalidates all the caches having a copy of the corresponding line and returns to the home location after traversing the ring.
4. After receiving the invalidation packet, the home location sends the invalidation acknowledgement to the requester, updates the directory, and unlocks the memory block.
5. The requester unlocks the cache line after receiving the invalidation acknowledgement packet.
end

Figure 5.2: Algorithm for processing upgrade request
Figure 5.3: State diagram for enforcing the sequential consistency
Shared-Read Request

begin
1. The requester (requesting cache controller) sends a shared-read request to the home location, and waits for the response.
2. if the block is in the shared state at the home location
   then
     (i) The home memory sends the cache line to the requester in multiple packets.
     (ii) The requester assembles the cache line, and
         if the line is in invalidation pending state
             then
                 it supplies the line to the processor (to service the read miss) and then changes the line state to invalid state.
             else
                 it supplies the line to the processor (to service the read miss), updates the cache, and then changes the line state to shared state.
     elseif the block is in the dirty state
         then
             (i) The home memory sends a shared-intervention request to the cache holding the dirty copy, call it dirty cache.
             (ii) The dirty cache sends a copy of the cache line in multiple packets as the intervention response to both the requester and the home location.
             (iii) The requester assembles the cache line, and
                 if the line is in invalidation pending state
                     then
                         it supplies the line to the processor (to service the read miss) and then changes the line state to invalid state.
                     else
                         it supplies the line to the processor (to service the read miss), updates the cache, and then changes the line state to shared state.
             (iv) After receiving the line from the dirty cache, the home location updates the memory block and the directory.
end

Figure 5.4: Algorithm for processing shared-read request
5.3 Protocol Implementation

The cache coherence protocol, described in Section 5.2, involves multicasting of the invalidation packet to all caches that have a copy of the line to be invalidated. Also, the home location must be locked until it receives the same invalidation packet. The routing scheme for the bidirectional ring, described in Section 3.1, allows efficient multicasting. However, there are two problems in using this scheme to multicast an invalidation packet.

First, it is necessary to detect that the invalidation packet has completed one traversal of the ring. This condition may be detected if all the routing bits are zero; which implies that every node should reset its corresponding bit(s) in the routing mask. A node cannot reset the node bit (a bit in the node segment), because the node segment is shared among all clusters; if the bit is reset then all nodes in the other clusters denoted by the same bit will miss the invalidation packet. Hence, only the cluster bits may be reset. The packet is removed from the ring by the node that finds all the bits in the cluster segment equal to zero. This problem occurs in the bidirectional ring because a single invalidation packet is circulated over the entire ring. In contrast, every destination local ring in the hierarchy receives its own copy of the invalidation packet; hence, resetting a node bit does not prevent the nodes in other local rings from receiving the invalidation packet.

Second, which node of a cluster may reset the cluster bit? If any node of the cluster that first sees the invalidation packet is allowed to reset the cluster bit, then all other nodes of the same cluster will miss the packet. Similarly, the last node of the cluster, considering a cluster to be an ordered set of nodes, cannot be allowed to reset the bit, because if it happens to be the first to receive the packet then all other nodes will miss it. We solve this problem by introducing a G-bit, which is a copy of the cluster bit of the most recently visited cluster. The first node of a cluster copies its own cluster bit into the G-bit, and then resets the cluster bit. The first node of a cluster is defined as the node whose upstream neighbor is a node of a different cluster. The packet is eventually removed by the node that detects all cluster bits and the G-bit equal to zero. A complete implementation algorithm is given in Figure 5.5. For the transmission of a non-multicast packet, the G-bit is set by the source node, and the destination node receives the packet when it finds the G-bit set and a matching routing tag. Thus, a zero G-bit consistently prevents the reception of the packet.

The example in Figure 5.6 illustrates the algorithm. The node (Rc0, N2) is the home location that issues the invalidation packet. The multicast address to invalidate copies of the cache line in nodes (Rc0, N3) and (Rc1, N0) is 11 1101. Bit 2 of the node segment is also set for the home location to receive the invalidation packet. In the Figure, an arrow attached to a
Implementation Algorithm
1. The home memory sets the G-bit and copies the directory entry into the routing mask of the invalidation packet. It also sets the bits in the routing mask that identify the home memory so that the home memory will also receive the returning invalidation packet. It then transmits the packet on the ring that is used to send the response to the requester.
2. When a node receives the invalidation packet, it performs the following action:
   if the node is the first node of a cluster
     then
       copy the corresponding cluster bit into the G-bit, and reset the cluster bit.
   if the G-bit is set
     then
       copy the packet into the receive buffer.
     elseif the G-bit and all cluster bits are equal to zero
       remove the packet.

Figure 5.5: Algorithm for multicasting invalidation on the bidirectional ring
packet pointing to the node shows that the packet is received by that node, whereas an outward arrow shows that the packet is removed from the ring. The G-bit is initially set by the home location, which causes node (Rc0, N3) to receive the packet. The packet then continues to node (Rc1, N0), which copies the corresponding cluster bit to the G-bit and resets the cluster bit. In this case the G-bit remains set causing the node to copy the packet to its receive buffer. The packet then continues along the ring and is copied by nodes (Rc1, N2) and (Rc1, N3). As the packet reaches node (Rc0, N0), the node resets the cluster bit 0 after copying it into the G-bit. The G-bit remains set causing the home location and node (Rc0, N3) to receive the packet. The packet is eventually removed by the node (Rc1, N0), because by the time it reaches this node all cluster bits will be zero and the G-bit is reset by the node itself.

![Figure 5.6: An example of invalidation packet flow](image)

Some downstream nodes of the home location within its cluster may receive the packet twice, for example node (Rc0, N3). This happens because when the packet enters the home location’s cluster its G-bit gets set causing all the nodes in the cluster whose corresponding bits in the node segment are set to copy the packet. The G-bit is finally reset by the first node of the next cluster, in this case node (Rc1, N0), which removes the packet from the ring. However, the invalidation packet reaching a node for the second time causes no ill-effect. Since, the home location arbitrates all subsequent requests, it can only respond to them after being unlocked.
by the same invalidation packet. Hence, any subsequent response from the home location must trail the invalidation packet. Thus, the invalidation packet that reaches a node the second time finds the line in the invalid state.
In this chapter a detailed analysis of simulation results is presented. In Section 6.1 the simulation environment and system model are described. The six benchmarks of the Splash2 benchmark suite [55] which are used in evaluating the performance in this chapter are described in necessary detail in Section 6.1.3. The analysis of simulation results is discussed in Section 6.2. First, the performance of the cache coherence protocol is discussed, then the network performance is evaluated. Finally, the overall performance of the benchmarks is discussed. The overall performance is measured in terms of speedup and program execution time.

6.1 Simulation Environment

System sizes of up to 64 processors have been simulated. To allow a direct comparison with the NUMAchine prototype [57], we assume that each node consists of four processors, a memory unit, an I/O unit and a network cache, all connected by a bus, as shown in Figure 6.1. Such a node is called a station. Each processor has a primary on-chip cache and a secondary off-chip cache. The network cache serves all the processors in the station, and stores only those cache lines that are fetched via the network. Inclusion is not guaranteed between the network cache and any secondary cache of a processor. A local ring connects up to four stations, and at most four local rings are connected by the global ring. Thus, a two-level ring hierarchy connects up to 64 processors, which is the largest system simulated. Table 6.1 shows the system parameters for our base configuration. All parameters of the base configuration are the same as those of NUMAchine [57]. The width of a link in a local ring and global ring are the same. A system using the bidirectional ring has been simulated by modifying the network interface to provide connection to the two constituent rings. The width of a link in a constituent ring of the bidirectional ring is the same as the width of a link in a local ring of the hierarchy.
Table 6.1: Base configuration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Processors</td>
<td>64</td>
</tr>
<tr>
<td>L1 cache size</td>
<td>16 KB</td>
</tr>
<tr>
<td>L2 cache size</td>
<td>1 MB</td>
</tr>
<tr>
<td>Network cache</td>
<td>4 MB</td>
</tr>
<tr>
<td>Cache line size</td>
<td>64 bytes</td>
</tr>
<tr>
<td>Processor clock</td>
<td>150 MHz</td>
</tr>
<tr>
<td>Bus clock</td>
<td>50 MHz</td>
</tr>
<tr>
<td>Ring clock</td>
<td>50 MHz</td>
</tr>
</tbody>
</table>
6.1.1 The Hierarchy

The network cache introduces a level of complexity to the cache coherence protocol. In this section, the changes in the basic protocol (described in Chapter 5) due to the network cache are described by defining only the additional states; the details of the basic protocol can be found in the NUMAchine technical report [57]. To maintain the cache coherence at both the network and station levels, a hierarchical, two-level directory exists. The directory is stored in SRAM located in the memory modules and the network caches. At the network level, the home memory maintains a full directory of routing masks for all of its cache lines.

The memory units and the network caches contain a valid/invalid (V/I) bit per cache line, which indicates whether the copy they have is valid. The network caches also contain a local/global (L/G) bit, which indicates whether the only valid copies of the cache line are on the local station. In the memory module, a separate L/G bit is not needed because this information is provided by the routing mask in the directory.

While three basic states (dirty, shared and invalid) are defined for the secondary cache in the standard way for write-back invalidate protocols, four basic states are defined for a cache line in a memory module or a network cache. The L/G and V/I bits are used to indicate the state of the cache line and can have the following meanings: local valid (LV), local invalid (LI), global valid (GV) and global invalid (GI). The LV and LI states indicate that valid copies of the cache line exist only on this station. In the LV state, the memory (or network cache) as well as some secondary caches have a valid copy. In the LI state, only one of the local secondary caches has a dirty copy. In GV, the memory (or network cache) has a valid copy of the cache line, and it is being shared by several stations, indicated by the routing mask in the directory. The meaning of the GI state differs slightly for the memory module and for the network cache. In both cases, the GI state means that there is no valid copy on this station. However, the GI state in the memory module also indicates that there exists a remote network cache (identified by the routing mask) with a copy in the LV or LI state. Each of the basic states also has a locked version. The locked versions are used to prevent accesses to a cache line while the line is undergoing some transition. Any requests for a cache line in a locked state are negatively acknowledged, and the requester will try again.

6.1.2 The Bidirectional Ring

For the bidirectional ring, the network cache maintains an invalidation pending state, in addition to the LV, LI, GV and GI states of the network cache. Consider a shared-read request issued by a processor to a remote home memory and assume that the line is in the GI state. The
network cache will lock the line in the GV-locked state (which corresponds to the shared-locked state in Figure 5.3; similarly the GI and GV states correspond to the invalid and shared states, respectively) before transmitting the request to the home memory. When the line is received by the network cache, it is also copied into the secondary cache of the requesting processor, and its state is changed to the GV state in the network cache. Next, consider the case of an invalidation outrunning the shared-read response. Upon the arrival of the invalidation request at the network cache, the state of the line is changed from the GV-locked state to the invalidation pending state. When the network cache receives the line, it sends the line to the requesting processor across the station bus, and then issues the invalidation over the bus in the next available bus cycle to invalidate the line in the secondary cache. The line is also invalidated in the network cache and its state is changed to the GI state. Thus, the invalidation pending state is introduced in the path from GV-Locked to GI states. All other transactions are performed the same as in the NUMAChine protocol [57].

6.1.3 Benchmarks

A number of benchmarks were selected from the SPLASH-2 suite, shown in Table 6.2, to investigate the comparative performance of the two networks - the hierarchy and the bidirectional ring. These applications were selected mainly because they show a variety of communication patterns and generate significant network traffic [55].

The following description of these benchmarks is mainly taken from the papers on Splash [48] and Splash-2 [55].

**FFT**

This is a kernel benchmark that solves a 1-dimensional FFT using the six step \( \text{radix} = \sqrt{n} \) algorithm. The data set contains \( 2^M \) points. The input data set has \( M \) equal to 16. Each point
is a complex number stored as a double. The points are stored in an \( N \times N \) array \( A \). Another major data structure is a \( \sqrt{N} \times \sqrt{N} \) array that is used to store the roots of unity. The data distribution is optimized for communication. Roots of unity are rearranged and distributed for only local accesses. Small sets of roots of unity elements are replicated locally at each node for 1D FFTs (fewer than root \( N \) elements are replicated at each node). Matrix data structures are padded to reduce cache mapping conflicts and to ensure that each processor's partition ends on a page boundary so that the data needed by the processor can be allocated to its local memory.

There are 6 major steps of 1D-FFT computation, which are as follows:

1. Matrix transpose: \( B \leftarrow A^T \).
2. Perform FFT computation on \( B \).
3. Update \( B \).
4. Matrix transpose: \( A \leftarrow B^T \).
5. Perform FFT on \( A \), and update \( A \).
6. Matrix transpose: \( B \leftarrow A^T \).

The processes are synchronized using barriers after steps 3, 5, and 6. The above six steps are repeated for backward FFT computation. The row-wise distribution of a matrix is performed so that each processor computes the FFT over a set of consecutive rows. In the three transpose phases, each processor reads a square sub-block of the source matrix, does a transpose, and writes into the destination matrix. The blocked transpose is used for cache-line reuse. Communication occurs in the three transpose phases which require an all-to-all communication. When a processor reads the sub-block, it generates read misses. Then, the cache lines are invalidated when the processor goes to write the sub-block in the destination matrix.

**Radix**

The radix sort is also a part of the NAS parallel benchmark. This is a kernel application that sorts a set of \( k \)-bit integer keys by examining \( r \) bits \( (r \leq k) \) in each iteration. The \( r \)-bit field is called a digit, where \( r \) is the log of the radix. In radix sort, the keys are progressively sorted one digit at a time. The keys are stored in a global array of integers. The sorted keys are stored in another global array. Both arrays are partitioned across the processors. The input contains 2M keys and 1K radix. The three steps of the parallel version of radix sort are:
1. Each processor reads the keys from its own partition of the input array, sorts them on the given digit, and creates its local histogram.

2. All local histograms are merged into a global histogram using prefix sum.

3. Each processor writes its own share of keys into another array in the sorted order. The keys in the sorted array are partially sorted up to the given digit.

The above steps are repeated as many times as there are digits in the keys. After each iteration the two arrays are alternately used, so that each processor reads its portion of keys from the partially sorted array of the previous iteration to perform sorting on the next digit. Both arrays are stored in a two dimensional array, where the first dimension index identifies the array. Thus, a copy operation is eliminated by simply switching the index. Barriers are used after every step to synchronize the processes.

All three steps of every iteration generate communication [38]. In step 1, each processor reads keys from the array which was written after sorting the keys in the previous iteration. Although the reading is sequential, a processor cannot independently start reading the keys unless all remaining processors finish writing into the sorted array in the previous iteration. Similarly, all processors in step 3 start writing into the sorted array at once after the new order of the keys is determined in the preceding step. Hence, the predominant pattern of communication in radix sort is bursty. The new location of keys, assigned to each processor, in the sorted array is random. This results in a significant amount of false sharing, because the next key in the same cache line could be written by a different processor. Hence, the spatial distribution of communication in step 3 is also random.

Ocean

This application simulates the ocean flow and calculates the eddy and boundary currents. The wind pressure and force caused by the friction of layers are also included in the computation. The ocean basin is simulated as a container. The computation is performed in a number of time steps. At each time step several independent calculations are made over a number of grids. Some calculations share the grid, whereas for others different grids are used. The resolution of a grid used in our study is $258 \times 258$. Hence, the main data structure is a number of two dimensional arrays which represent different grids. In a few cases dimensionality is increased. The major computation involves solving differential equations over grids. The continuous differential equations are transformed into discrete differential equations and then solved using the multigrid SOR method.
The partitioning of computation over all processors is done by decomposing the data domain. Thus, each processor performs computation over its assigned sub-domain which consists of few grid points. The exchange of data takes place only for the boundary elements. This shows a near-neighbor communication pattern. The data written by local processors in one step are read by remote processors in the subsequent step. Thus, the sharing pattern allows optimization by the cache coherence protocol and prefetching.

Water Nsquared

This application computes forces and potentials in a system of water molecules. The computation is performed over a user-specified number of time steps. The number of time steps is chosen to make the system reach a steady state. In every time step, Newtonian equations of motion for water molecules in a cubical box with a periodic boundary condition are solved. The inter-molecular forces are computed to evaluate the interaction of molecules, and the intra-molecular forces are computed to evaluate the forces that cause the motion of atoms within a molecule. The total potential is computed as the sum of intra- and inter-molecular potentials. At the end of every time step, new positions and velocities of molecules are computed. The forces and potentials are computed using $O(n^2)$ algorithm, and a predictor-corrector method is used to integrate the motion of the water molecules over time.

The main data structure is an array that stores the information about all molecules. The size of the array is equal to the number of molecules in the system. The information about a molecule is stored in a structure composed of two arrays - one linear array $VM$ of three elements, and a three dimensional array, called $F$ array. The three elements of the $VM$ array store the location of the molecule's centre of mass. The $F$ array stores the values for each atom in the molecule, and is indexed on the displacement and its six derivatives in one dimension, three spatial directions in the second dimension, and the number of atoms in the third dimension. The global (shared) address space stores the main array and a global structure that is used to keep several locks per processor. Every element of the main array is 672 bytes in size. The size of the main array for the input file containing data for 512 molecules is about 350 Kbytes. Apart from the shared space, every processor maintains its own private space whose size is 72 times the number of molecules per processor in addition to 720 bytes.

There are numerous independent computations in every time step that are organized as separate tasks. The computation of inter-molecular forces is the most time consuming task, as it takes $O(n^2)$ time units to execute. Parallelism is available within and across the tasks. Every task is distributed over all processors. The main array is partitioned across the processors such that each processor gets a share of molecules which are contiguous in the array (not necessarily
physically contiguous). The communication is phase structured, that is the remote reads are performed by processors in the inter-molecular computation phase, and the array is updated in the next phase. Every processor keeps a private data-space to store the results of computation and then updates the global memory once. The synchronization involves locks and barriers.

**Cholesky**

This is a kernel that performs parallel Cholesky factorization of a sparse matrix. The object is to calculate a lower triangular matrix $L$ of a given definite matrix $M$ such that $M = LL^T$, where $L^T$ is the transpose of $L$. The Cholesky factorization involves three steps: reordering, symbolic factorization, and numerical factorization. This program only performs numerical factorization which is the most time consuming among all steps. The principal data structure is a representation of a sparse matrix, as given below:

```c
typedef struct {
    int n, m, *col, *startrow, *row;
    double *nz;
} SMatrix;
```

The integers $n$ and $m$ are the number of columns and the number of non-zero elements of the matrix, respectively. All non-zero elements of each column are stored contiguously in the array nz. The input matrix is $11948 \times 11948$.

The supernodal elimination method is used to perform numerical factorization. A supernode is a set of columns having nearly identical non-zero structures. In a column-oriented sparse Cholesky factorization, a column modification is performed by adding a multiple of a column into another column to cancel a non-zero element in the upper triangle. This operation is replaced by supernodal modification in supernodal Cholesky factorization. In supernodal modification a column is modified by all the columns of a supernode at once. For each supernode a count is kept that tracks the number of modifications to be performed on the columns of that supernode. When the count reaches zero, the supernode is put in a global task queue. A supernode in the global task queue is ready to be picked up by any processor to perform supernodal modification using this supernode. The data-sharing pattern is quite random. A supernode may be modified by several processors before it is picked up by one processor to modify other supernodes. After a supernode has completed its share of modifications to all other supernodes, it is no longer referenced by any processor.
Ray-tracing

This application program is used to render a three dimensional scene using a ray tracing algorithm [47]. In ray tracing, rays are projected from viewpoints through the pixels on the image plane to the scene to be rendered, and these rays are traced. Some of these rays are reflected back, thus generating a tree of secondary rays. The ray tracing ends when either the rays leave the scene or some user provided criterion is met; an example of the user specified criterion could be the maximum number of levels allowed in the ray tree. The opportunity for parallelism is found in the ray tree which can be distributed across the processors. The scene database is read-only, whose single copy can be distributed among the processors using a round-robin page allocation. A hierarchical uniform grid is used to represent the scene and early ray termination is used to terminate the ray tracing. The image plane is partitioned among processors as contiguous blocks of pixel groups. The task queue is distributed among the processors and task stealing is used for load balancing. The input scene used for the simulation is a car on a checkerboard floor. The image plane has 512 x 512 pixels, and the data set size is about 10 Megabytes. The access pattern of the raytrace is unstructured and it shows mediocre spatial locality [55]. The reason for low spatial locality is that the access pattern to the read-only scene database is highly unstructured, one field of a voxel or a polygon may be touched by one processor, while the next field may be touched by a different processor.

Cache Miss Profile

In Figure 6.2, the secondary cache miss ratios are shown against different line sizes for each of these benchmarks. Three benchmarks - FFT, Radix and Ocean - were used to study the impact of the cache coherence protocol on the latency of remote transactions for the bidirectional ring. These benchmarks were simulated for a 64-processor system using different cache line sizes, ranging from 32 bytes to 128 bytes. The cache size remains constant in these simulations.

The non-white portion of each bar indicates the misses serviced by either remote network caches or remote memories. The white portion shows the misses serviced locally, either by the local network cache or the local memory. Each portion of a bar shows the percentage of secondary cache accesses that cause misses in that category, hence the total bar value gives the secondary cache miss ratio. FFT and Ocean show a significant reduction in the miss ratio when the cache line size is increased, mainly due to the extensive spatial locality found in these two applications. For Radix, the miss ratio increases when going from the 64-byte line size to the 128-byte line size, due to the false sharing which becomes significant for large cache lines. This is evident by the increase in intervention misses for the 128-byte line size. These statistics
are consistent with the miss profile of the same benchmarks on 32 processors reported in the Splash-2 paper [55]. We note that there the remote misses are not categorized, as done in Figure 6.2, for detailed analysis.

![Breakdown of Cache Misses](image)

Figure 6.2: Secondary cache miss profile

The TPC-B and TPC-D benchmarks show similar cache miss ratios [32, 54]. These benchmark applications are part of the Transaction Processing Performance Council's (TPC) benchmark suite for commercial applications [53]. The TPC-B represents Online Transaction Processing (OLTP) workloads. The TPC-D is an industry-standard benchmark used to represent relative performance for 17 real-world Decision Support System (DSS) queries. The commercial database market is dominated by OLTP and DSS applications, hence, these benchmarks represent this market segment. Trancoso et al [54] show through a simulation-based study that the primary cache miss ratios are 5.5%, 3.4%, and 4.8% for queries Q3, Q6, and Q12 of TPC-D benchmark respectively. Their respective secondary cache miss ratios are 0.8%, 0.6%, and 0.5% [54]. The application profiles of Q6 of TPC-D and TPC-B show that the secondary cache miss ratios are 0.18% and 2.2% respectively [32]. The miss ratios were obtained from the data collected by running these benchmarks on Sequent Symmetry 2000 and 5000 SMP systems [33, 45] using event counters and logic analyzers. Nearly 50% of the secondary cache misses are satisfied by the local memory and about 25% are found in the network cache. Thus,
about a quarter of secondary cache misses result in generating the network traffic. In designing the interconnect for the NUMA-Q Architecture, Sequent ran many tests using Oracle and Informix applications [46]. These tests showed that over 51% of the secondary cache misses are found in the local memory, and an additional 30% are found in the network cache. Thus, the remaining 19% may require remote memory access through the interconnect.

The statistics of memory references gathered for commercial database applications demonstrate good data locality, which is identical to the behavior of the sample benchmarks chosen for this performance study. Hence, the performance results will likely be applicable for scientific as well as database applications.

6.2 Results

6.2.1 Cache Coherence Protocol Performance

We use the latency of remote transactions as a measure to evaluate the performance of the cache coherence protocol. The latency is measured in processor clock cycles. In the simulated system the processor clock is three times faster than the ring (backplane) clock. The components of latency that involve local processing, such as interacting with the ring interface, memory access, etc, are the same for both systems. Thus, the major source of difference is the network component. Figures 6.3, 6.4, and 6.5 show the latency of both networks for FFT, Radix, and Ocean, respectively.

Transactions that involve invalidation are exclusive reads and upgrades. We observed that the latencies of both of these transactions for all three benchmarks in the bidirectional ring are lower than in the hierarchy. This shows that invalidations are performed in the bidirectional ring with no substantial overhead. It is important to note that the invalidation of a cache line in the bidirectional ring starts immediately after the packet leaves the home memory station; whereas in the hierarchy, for example if the sharing nodes are scattered, the invalidation can be delayed until the invalidation packet first goes up to the global ring and then comes down to the local rings. Hence, the relative performance of upgrades improves in the bidirectional ring for the line size that leads to high sharing, such as for the 128-byte size. One shortcoming of the cache coherence protocol in the bidirectional ring is that the invalidation packet traverses the whole ring before the requester gets an acknowledgement to unlock itself, whereas in the case of the hierarchy the requester unlocks itself as soon as it receives the invalidation packet, that is while other caches are being invalidated. When sharing is low, such as for smaller line sizes, the difference in the latency of upgrades in the two networks is marginal.

The statistics of per processor shared reads that are affected by the intervening invalidations
Figure 6.3: Transaction latency vs. line size for FFT

Figure 6.4: Transaction latency vs. line size for Radix
in the bidirectional ring cache coherence protocol is given in Table 6.3. The table shows that the majority of applications show a very small number of affected shared reads. But, in the Ray trace application 11.93\% of shared reads are affected. This shows that a significant number of shared reads may be affected in some cases, which justifies the cost of including a pending invalidation state to improve the performance as opposed to aborting the affected shared reads.

### 6.2.2 Network Performance

Network utilization is the metric used to gauge the network performance. Figures 6.6, 6.7, and 6.8 show the network utilization for FFT, Radix, and Ocean, respectively, for three different cache line sizes. We observe that the ring utilization never exceeds 25\% for the bidirectional ring, while for the hierarchy it goes up to 45\%. This shows that, on average, the communication traffic for these benchmarks does not stress the network. When utilizations of the constituent rings are compared, the hierarchy shows a significant difference between the utilization of the global and local rings, whereas for the two rings of the bidirectional ring they are essentially equal. The local ring shows a higher utilization than the utilization of either of the two rings of the bidirectional ring. The global ring consistently shows almost double the utilization of the local ring.
Table 6.3: Statistics of affected shared reads

<table>
<thead>
<tr>
<th>Application</th>
<th>Line size (bytes)</th>
<th>No. of remote shared reads</th>
<th>No. of affected shared reads</th>
<th>% of affected shared reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT</td>
<td>32</td>
<td>2857</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FFT</td>
<td>64</td>
<td>1416</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>FFT</td>
<td>128</td>
<td>955</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Radix</td>
<td>32</td>
<td>5122</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Radix</td>
<td>64</td>
<td>2640</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Radix</td>
<td>128</td>
<td>1665</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>Ocean</td>
<td>32</td>
<td>17847</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>Ocean</td>
<td>64</td>
<td>14275</td>
<td>114</td>
<td>0.8</td>
</tr>
<tr>
<td>Ocean</td>
<td>128</td>
<td>14203</td>
<td>666</td>
<td>4.69</td>
</tr>
<tr>
<td>Ray trace</td>
<td>64</td>
<td>14587</td>
<td>431</td>
<td>2.95</td>
</tr>
<tr>
<td>Ray trace</td>
<td>128</td>
<td>10429</td>
<td>1244</td>
<td>11.93</td>
</tr>
<tr>
<td>Water Nsquared</td>
<td>64</td>
<td>3501</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Cholesky</td>
<td>64</td>
<td>14114</td>
<td>84</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 6.6: Network utilization vs. line size for FFT
Figure 6.7: Network utilization vs. line size for Radix

Figure 6.8: Network utilization vs. line size for Ocean
The average ring access delay for the hierarchy is shown in Figures 6.9 and 6.10. The global ring access delay is the delay experienced by a packet moving from a local ring to the global ring while passing through the buffer at the inter-ring interface. A local ring access delay is the delay experienced by a packet from the time it is transferred over the station bus, from the processor to the NIC, to the time it is transmitted on the ring. The ring access delay is computed for a local ring of the hierarchy, and for each constituent ring of the bidirectional ring. For large cache-line sizes, such as the 128-byte size, owing to increase in sharing, the traffic increases at both levels of the hierarchy. This causes the large delays in Figures 6.9 and 6.10.

Radix is characterized by bursty communication and sharing of writable data. False sharing increases when a 128-byte cache line size is used, which causes a large number of cache line transfers across the network. The large cache line size also increases the volume of communication. By increasing the line size from 64 to 128, the number of packets transferred per cache line is doubled. Hence, the sizes of receive and transmission buffers in the ring interface, and of up and down buffers in the inter-ring interface are scaled up accordingly. The cumulative effect of burstiness and high volume of communication appears as high network utilization and large ring access delays, for both the global and local rings. The high global ring access delay causes the flow control mechanism to lock the local rings. The simulation statistics show that the local rings are locked many times. This indicates that the flow control mechanism which is proposed for the hierarchy is effective in handling the buffer overflow problem at the inter-ring interface caused by bursty traffic. But, the locking of local rings increases the latency of the affected transactions.

Figures 6.11 and 6.12 show the average access delays of the two constituent rings of the bidirectional ring. As expected from the comparable utilizations of the two rings, their access delays are also comparable. Notice that the bidirectional ring does not show an unusually high delay for Radix, which means that its handling of bursty traffic is better. Hence, the bidirectional ring is expected to provide better support for block transfers, for example due to page migration, which is a major source of bursty communication.

### 6.2.3 Overall Performance

The speedup is the simplest measure to gauge the overall performance of a system. We use this measure to see the impact of a network, specifically the bidirectional ring, on the overall performance. The speedup is measured by using parallel execution time. Parallel execution time for the SPLASH-2 benchmarks is the time from when threads are created until the time the wait() system call is executed. For different system sizes the hierarchy is constructed in
Figure 6.9: Average local ring access delay (in ring clock cycles)

Figure 6.10: Average global ring access delay (in ring clock cycles)
Figure 6.11: Average ring-0 access delay (in ring cycles)

Figure 6.12: Average ring-1 access delay (in ring cycles)
such a way that no more than four nodes are connected in any ring. For instance, a single ring is used to connect up to 16 processors (four stations), and two local rings are used to connect up to 32 processors (eight stations) and so on. In the case of two local rings a global ring is used to connect them; a single global ring is used to connect up to four local rings which is the largest system size we have simulated. Since most of the benchmarks of the Splash-2 suite do not generate a high volume of communication to make the speedup sensitive to the network performance, we cannot expect the bidirectional ring to contribute to a large speedup. Figures 6.13, 6.14, and 6.15 show the speedup curves for all the benchmarks given in Table 6.2. These benchmarks exhibit a variety of communication patterns and they have relatively higher communication to computation ratio [55].

![Speedup vs. system size (in no. of processors) for Ocean and Water](image)

**Figure 6.13: Speedup vs. system size (in no. of processors) for Ocean and Water**

These Figures show that the bidirectional ring provides either better or comparable speedup than the hierarchy as the system size is increased. The difference in improvement is also more significant for the benchmarks showing a high volume of communication, such as FFT and Radix.

To examine the impact of network performance on the overall performance of the application, Table 6.4 shows execution times of FFT, Radix, and Ocean for three different cache line sizes. By increasing the cache line size, the volume of communication may increase due to false sharing and the increase in the number of packets required to transfer a cache line, which
Figure 6.14: Speedup vs. system size (in no. of processors) for FFT and Radix

Figure 6.15: Speedup vs. system size (in no. of processors) for Cholesky and Ray trace
Table 6.4: Parallel execution times (in ms)

<table>
<thead>
<tr>
<th>Cache line size</th>
<th>FFT</th>
<th>RADIX</th>
<th>OCEAN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bidirectional</td>
<td>Hierarchy</td>
<td>Bidirectional</td>
</tr>
<tr>
<td>32</td>
<td>25.21</td>
<td>30.35</td>
<td>303.14</td>
</tr>
<tr>
<td>64</td>
<td>18.57</td>
<td>25.04</td>
<td>297.79</td>
</tr>
<tr>
<td>128</td>
<td>18.73</td>
<td>26.02</td>
<td>409.48</td>
</tr>
</tbody>
</table>

may increase the execution time. But, large cache line sizes may improve performance too due to spatial locality. However, the overall performance depends upon the combination of the above factors. Since the two systems are essentially the same except for the difference in their interconnection network – hierarchy for one and for the other bidirectional ring – any difference in the performance by changing the line size can be attributed to the performance of the interconnection network. It is evident from Table 6.4 that the execution time for a system using the bidirectional ring is lower than for the hierarchy. The difference between their execution times increases when larger line size is used. For instance, the percentage difference in the execution time of the bidirectional ring and the hierarchy increases from 17% to 28% for FFT, from 8% to 30% for Radix, and from 9% to 28% for Ocean, as the line size is increased from 32 bytes to 128 bytes.

In the above performance evaluation, the bidirectional ring was simulated to be running at 50 MHz, the same speed as that of the hierarchy; as a result it has more bandwidth than the hierarchy, which may tilt the comparison in its favor. To remove this unfairness to the hierarchy, we performed two other types of simulations. In one case, we decreased the clock rate of the bidirectional ring, and in the other case we increased the clock rate in the hierarchy. In Figure 6.16, normalized execution times are plotted for four different configurations of the hierarchy and the bidirectional ring. In two configurations of the hierarchy, 100 and 50 MHz ring clock speeds are considered. For the two configurations of the bidirectional ring, 50 and 25 MHz ring clock speeds are assumed. All execution times are normalized to the execution time for the 50 MHz hierarchy. In comparing the performance of the hierarchy and the bidirectional ring running at half the clock speed of the hierarchy, the hierarchy enjoys the advantage of more bandwidth, hence the performance estimates for the bidirectional ring are conservative.

By reducing the clock speed from 50 to 25 MHz, the execution time for the bidirectional ring is increased. But, it remains below the execution time for the hierarchy running at 50 MHz clock for all benchmark applications except Ray trace. This shows that: 1) the performance of these applications is more sensitive to latency than to bandwidth, and 2) the bidirectional
ring exhibits lower latency due to better exploitation of communication locality. It is also interesting to compare the execution time for the 100 MHz hierarchy and 50 MHz bidirectional ring. The comparison shows no significant performance gain by increasing the ring clock speed for the hierarchy, which essentially increases its bandwidth at greater cost and complexity of implementation. For most of the applications the execution times remain close to those for the 50 MHz bidirectional ring.

![Normalized execution time for the hierarchy and the bidirectional ring running at different clock speed on 64-processor system](image)

Figure 6.16: Normalized execution time for the hierarchy and the bidirectional ring running at different clock speed on 64-processor system

**Summary**

The relative performance of the bidirectional ring and a hierarchy of unidirectional rings was evaluated by running benchmark applications from the Splash-2 suite on simulated systems based on the two networks. The system parameters were analogous to the NUMAchine prototype. The bidirectional ring shows a significant reduction in latency, as much as 50%, which indicates that the proposed cache coherence protocol is an efficient scheme. We also measured the network utilization, which is a more direct indicator of the network performance. The bidirectional ring tends to balance the load over the two constituent rings, showing not more than
5% difference in the utilization of the two rings, whereas the hierarchy shows a significantly large discrepancy in the utilization of local and global rings. The global ring usually shows double the utilization of the local rings. We also measured the overall performance of several benchmarks with different communication characteristics and communication to computation ratios. Although no benchmark of the Splash-2 suite loads the network to a point of significantly affecting the execution time of the application, speedup curves show some gains for the bidirectional ring. Larger gains are observed for large cache line sizes, which increase both sharing and the volume of network traffic.
CHAPTER 7
Reliability

The bidirectional ring offers potential for increased reliability. It can survive the failure of some node or of one ring. In the case of failure of one ring, the other ring may carry all communication, though with lower performance. Its ability to handle faults by gracefully degrading the performance is desirable for applications that require high availability, such as on-line transaction processing.

A bidirectional ring can tolerate a single link failure or a single node failure. It can also tolerate multiple link failures if all the failed-links are on the same ring. The fault handling capability of a bidirectional ring is contingent on: (i) successful detection of faults, and (ii) recovery after failure. In Section 7.1, different modes of failure are discussed. A solution for bypass mode of failure and its implementation in the network interface is also discussed. The fault detection and recovery procedures are discussed in Section 7.2. The bypass mode is interesting because it also increases the repairability of the system [23]. A cache coherence protocol that handles the bypass mode of failure and its implementation is discussed in Section 7.3. Finally, the performance of a faulty system is evaluated and the results are presented in Section 7.4.

7.1 Modes of failure

There are two modes of failure. One is the ring-down mode and the other is the bypass mode. In the ring-down mode, one ring is shut down after switching all the traffic to the other ring, as shown in Figure 7.1. This mode of failure is caused by link failures. The performance degradation in a recovered system is expected to be high, because all the load is shifted to one ring.

In the bypass mode of failure, the bidirectional ring is essentially cut apart at one point and the rings are wrapped around at the two ends, as shown in Figure 7.2. This is realized by switching the traffic from one ring to the other ring at the two end points. In the case of a single node failure, the two nodes adjacent to the failed-node divert the traffic from the ring
that leads to the failed-node to the other ring. For example, node (Rc1, N1) switches the traffic on the counter-clockwise ring to the clockwise ring, and node (Rc1, N2) performs vice versa. The bypass mode allows hot swapping, as a node can be removed for repair or upgrade without disrupting the normal machine operation; it enhances the repairability of the system.

A sample network interface design that allows bypass is shown in Figure 7.3. There is a pair of identical bypass control circuits, one for each direction. The bypass controller controls the demultiplexers (DEMUX) at the receiving end and the multiplexers (MUX) at the transmitting end. There are two control signals, called next-down signals, one for each direction, which indicate that the next downstream node in the respective direction has failed. The signal is set during the recovery phase after detecting the fault. When the signal is set, all the packets that are supposed to be forwarded to the next downstream node will be bypassed to the ring of opposite direction. The controller also arbitrates between the transmission and the bypass buffers. In the Figure, the signal for-this-node indicates that the incoming packet is to be received, which is set after decoding the destination address.

The transmission and bypass buffers can be combined into a single buffer with two write ports and one read port. This will simplify the controller by eliminating the arbitration circuitry. The performance difference between the combined buffer scheme and the separate buffer scheme is marginal, as discussed in Section 7.4.
7.2 Fault Detection and Recovery

Detecting a fault in a high speed network is a non-trivial task. The procedure to detect a fault mainly depends upon the type of fault. An intermittent fault is hard to discover, as opposed to a permanent fault. In this section we will focus on detecting a subset of permanent faults.

A permanent fault can corrupt the packet in several ways. A control bit error may cause the loss of packets. An error in the data part of the packet could be fatal, and its detection needs higher level support. If the routing bits are corrupted, the packet may be misdirected to a non-destination node, or may be lost and circulate on the ring forever. A garbage collection scheme [5] is usually employed to remove wandering packets. A stuck-at-0 fault at the busy-bit may make the slot empty, causing the packet to be lost. In the remainder of this section two schemes are discussed, one for garbage collection, and the other for recovery from a busy-bit error.

A garbage collection scheme is employed to detect and remove the packets that circulate on the ring forever. In a simple scheme, a node is designated to be the garbage collector, and a control bit, called garbage-bit, is introduced to identify such packets. When a sender transmits the packet it resets the garbage-bit. The garbage-bit is set by the garbage collector node when it observes the packet the first time. A packet whose garbage-bit is equal to one is removed by the garbage collector node. The garbage collection scheme is discussed in conjunction with the reliable cache coherence protocol in Section 7.3.

A stuck-at-0 fault that affects the busy-bit can be detected by the node that is waiting for
Figure 7.3: Network interface to perform bypassing
the response. Whether the packet contains a request or response, the requester can always
detect the packet loss using a time-out mechanism. A time-out counter can be introduced that
keeps track of the number of times the time-out period elapses for a packet. After a certain
count, the node propagates a control packet to halt all the nodes. When a node receives the
halt packet, it stops generating its own requests and gradually comes to a halt after serving
the requests in the pipeline. Then, a fault detection and recovery procedure can be initiated to
isolate the faulty link.

While the stuck-at-0 fault affecting the busy-bit is a non-destructive fault, the stuck-at-1
fault affecting the busy-bit could be destructive. When a packet is removed from the ring by
a node, the node simply resets the busy-bit making the slot empty. Since, other information
including the routing tag remains valid in the slot, a stuck-at-1 fault that sets the busy-bit
causes the slot to become a non-empty slot again. The destination node(s) that receives this
slot considering it a valid slot may repeat the transaction that could be destructive, such as
a write. The invalidation protocol proposed for the bidirectional ring in Section 5.3, however,
ensures that an invalidation packet that has been removed from the ring will not cause the
repetition of the invalidation transaction if the busy-bit of the packet is subsequently affected
by a stuck-at-1 fault. The invalidation packet is removed when all the cluster bits are equal
to zero, hence even if the busy-bit is set later, the packet will not be received by any node. A
garbage collector will eventually remove such packets. This leads us to conclude that if a node
while removing the slot not only resets the busy-bit, but also resets all the cluster bits in the
routing tag, then the stuck-at-1 fault can be handled in the manner discussed above.

7.3 Reliable Cache Coherence Protocol

The algorithm to implement the cache coherence protocol on a bidirectional ring, described
in Section 5.3, can easily be adapted to work in the bypass mode of failure. There are three
issues that need to be addressed before suggesting any modifications in the algorithm. First,
when a packet is switched to another ring, then some of the nodes of the most recently visited
cluster may remain to be invalidated. This may happen because the cluster is split by the
break, leaving the nodes at both end points. Hence, the remaining nodes will be found in
the last cluster before the packet again reaches the point of switching the ring. It means
that information about partial invalidation of a cluster needs to be saved. Second, how is the
condition for removing the packet from the network detected? Third, the home memory may
receive the invalidation packet before all copies are invalidated. In this case, it unlocks the
home location and sends an acknowledgement to the requester. Thus, the requester may be
unlocked before all copies in the network have been invalidated. This may lead to a violation of sequential consistency.

The first two issues can be resolved by introducing simple bypass logic that swaps the G-bit and the cluster bit of the packet before switching it to another ring. As a result, the cluster bit acquires its original value, whereas the G-bit becomes zero. The zero value of the G-bit prevents any node of the recently visited cluster from copying the invalidation packet again. On the other hand, the original value of the cluster bit will allow the remaining nodes of the cluster to be invalidated. All cluster bits are zero when the packet again reaches the point of switching rings, which indicates that the packet has completed one traversal of the ring. The bypass logic detects this condition and removes the packet instead of switching rings. Notice that the packet may not be removed erroneously at the first instance of switching the ring, because all the cluster bits are not yet zero. At least the cluster bit for the home location's cluster remains set at this point. In summary, the bypass logic removes the packet when it finds that all cluster bits are equal to zero, otherwise it swaps the G-bit and the cluster bit and stores the packet in the bypass buffer to be later transmitted on the other ring. It is important to note that the additional logic that controls the flow of invalidation packets in the bypass mode does not affect the normal communication path; it is concentrated in the bypass path.

The above scheme ensures sequential consistency. The sequential consistency is guaranteed if the home location is locked from the time the invalidation packet is issued until all copies of the line are invalidated, as discussed in Chapter 5. Hence, the requester remains locked during this period as well. Thus it is ensured that both copies - the old and the new copies of the line - are not visible to some nodes at the same time. This means that to violate the sequential consistency not only the requester must be unlocked before all the copies are invalidated, but there must exist at least one alternative path between the cache, whose line is to be invalidated by the invalidation packet that is in transit, and the requester. In the case of bypass mode, the bidirectional ring is practically converted to a large single ring. Hence, there is no alternative path for the two nodes to communicate. Thus, sequential consistency is maintained by the above protocol.

We will first explain the flow of an invalidation packet in a good network, and then in the faulty network. The example in Figure 7.5 illustrates the good case. The faulty case is shown in Figure 7.6. The node \((Rc0, N2)\) is the home location that issues the invalidation packet. The multicast address to invalidate copies of the cache line in nodes \((Rc0, N3)\) and \((Rc1, N0)\) is 11 1101. Bit 2 of the node segment is also set for the home location to receive the invalidation packet. In the Figure, an arrow attached to a packet pointing to the node shows that the packet is received by that node, whereas an outward arrow shows that the packet is removed from the
Implementation Algorithm

1. The home memory sets the G-bit and copies the directory entry into the routing mask of the invalidation packet. It also sets the bits in the routing mask that identify the home memory so that the home memory will also receive the returning invalidation packet. It then transmits the packet on the ring that is used to send the response to the requester.

2. When a node receives the invalidation packet, it performs the following action:
   if the node is the first node of a cluster then
copy the corresponding cluster bit into the G-bit.
if the G-bit is set then
copy the packet into the receive buffer.

3. If the G-bit and all cluster bits are equal to zero then
   remove the packet from the network.
else
   if the next downstream node has failed then
   if all cluster bits are equal to zero then
   remove the packet from the network.
   else
   swap the cluster bit and the G-bit.
copy the packet into the bypass buffer.
else
   forward the packet to the next downstream done.

Figure 7.4: Algorithm for multicasting invalidation in the bypass mode
ring. The G-bit is initially set by the home location, which causes node (Rc0, N3) to receive the packet. The packet then continues to node (Rc1, N0), which copies the corresponding cluster bit to the G-bit and resets the cluster bit. In this case the G-bit remains set causing the node to copy the packet to its receive buffer. The packet then continues along the ring and is copied by nodes (Rc1, N2) and (Rc1, N3). As the packet reaches node (Rc0, N0), the node resets the cluster bit 0 after copying it into the G-bit. The G-bit remains set causing the home location and node (Rc0, N3) to receive the packet. The packet is eventually removed by the node (Rc1, N0), because by the time it reaches this node all cluster bits will be zero and the G-bit is reset by the node itself.

Figure 7.5: An example of invalidation packet flow in a good network

In Figure 7.6 a link-failure is introduced between the nodes (Rc1, N1) and (Rc1, N2) to illustrate the bypass mode. This causes the traffic on the counter-clockwise ring to be diverted to the clockwise ring at node (Rc1, N1), and vice versa at (Rc1, N2). The invalidation transaction is started as normal by the home location (Rc0, N2) without prior knowledge of any fault in the network. When the invalidation packet reaches node (Rc1, N1), it is transferred to the clockwise ring. Since all the cluster bits of the packet are not equal to zero, the cluster bit-1 is swapped with the G-bit. Thus, the cluster bit-1 becomes 1 again. Although the corresponding cluster bit is 1 while the packet starts traversing the clockwise ring from node (Rc1, N1), the zero value
of the G-bit prevents node (Rc1, N0) from copying the packet again. The packet traverses the ring in the normal way up to node (Rc1, N2), where it comes across a point of switching the ring again. Since at this time all the cluster bits of the packet are zero, the bypass logic removes the packet from the network, instead of switching the ring.

![Diagram of packet flow in a faulty network]

Figure 7.6: An example of invalidation packet flow in a faulty network

### 7.3.1 Garbage Collection

The garbage collector node removes a wandering packet when the packet revisits the node. If the garbage collector belongs to the same routing cluster as that of the home location, then the invalidation packet may be removed while the G-bit is still set. For instance, if node (Rc0, N3) is the garbage collector in the above example, it will remove the invalidation packet when the packet revisits the node, marked by an asterisk in Figure 7.5. Since the G-bit remains set when the packet is removed, a stuck-at-1 fault at the busy-bit may cause some of the nodes to receive the packet. This shows that only the condition that all cluster bits are zero does not guarantee safety from the stuck-at fault of the busy-bit. Safety can be achieved if the condition that the G-bit must be zero is also added. Hence, the problem can be resolved if a node that removes the packet is also required to reset the G-bit.

The above garbage collection scheme fails on a faulty ring that is running in the bypass
mode. To illustrate the problem, consider \((Rc0, N3)\) to be the garbage collector node. When the packet revisits the node, its garbage-bit is found set and the packet is removed by the node; the packet is marked by an asterisk in Figure 7.6. In this case a good packet is removed, which will break down the cache coherence protocol. The problem occurs because the packet that is bypassed once has not yet completed a ring traversal. The packet must have switched the ring twice to complete one traversal of the ring. Therefore, a wandering packet that has switched the ring twice may be removed from the ring. This condition can be detected by introducing a bypass-bit. The bypass-bit is reset by the source node, and is toggled by the bypass logic. The packet whose bypass bit is set is the packet that has undergone bypass once, hence even if its garbage bit is set it should not be removed from the ring. The garbage collector removes the packet only when the garbage-bit is set but the bypass-bit is zero. This condition is trivially satisfied on a good ring, because the bypass-bit remains zero as it is never toggled there.

### 7.4 Performance Evaluation of a Faulty System

A faulty bidirectional ring system was simulated to evaluate the performance degradation caused by a fault. Different faulty networks were used to simulate different modes of failure. A single ring system was used to accurately simulate the ring-down mode of failure. The bypass due to link failure can be simulated by implementing the bypass logic and diverting the traffic at a pair of adjacent nodes. Some benchmark programs need a power-of-two number of processors to run, for example FFT, hence to keep the simulation of benchmarks simple, the bypass caused by a node failure was not simulated. In addition to the transmit buffer that stores the packets originated at the node, a bypass buffer is used under the control of bypass logic to store the packets that are switching rings. The bypass logic also acts as an arbiter to select the packet from either of the two buffers for actual transmission. A simple arbitration scheme was simulated that always selects the packet from the bypass buffer as long as the buffer is not empty. We call this separate buffer scheme \textit{Faulty-SB}. Alternatively, the transmit and bypass buffers can be unified into a single buffer. This saves time and resources that are otherwise spent on arbitration. The unified buffer scheme is called \textit{Faulty-UB}. The remaining features of the system are the same as described in Section 6.1. Only 64-processor systems were simulated, because the communication overhead is more conspicuous for this system size.

#### 7.4.1 Network Utilization

The network utilization is a direct measure for assessing the performance of a network. The network utilization graphs for all six benchmark programs are shown in Figures 7.7 to 7.12.
The utilization of the single ring simulating the ring-down mode is expected to be high, because the traffic of two rings is carried by a single ring. Indeed the utilization for a single ring is by far the highest in all graphs.

The constituent rings of the faulty bidirectional ring for both Faulty-SB and Faulty-UB, that simulate the bypass mode, show comparable utilization, maintaining the load balancing trend of a good bidirectional ring. To understand the load balancing achieved on the faulty ring, consider a node that communicates with a set of nodes through the clockwise ring on a good network, but which are not reachable through the clockwise ring due to a fault. All packets destined to the nodes of this set now switch to the counterclockwise ring to reach the destinations. Similarly, in the return path, the response packets which normally travel on the counterclockwise ring, must now switch to the clockwise ring to reach the given node. As long as for each pair of nodes the number of requests generated by both nodes for each other remains comparable, the load on each ring will tend to be balanced.

The second trend which is evident from these graphs is that there is no significant difference between the utilization of Faulty-SB and Faulty-UB rings. This is because both structurally and behaviorally the two faulty rings are identical. The arbitration scheme has little bearing on the utilization of the network. It is also evident from these graphs that for most of the applications the difference between the utilisations of a good ring and a faulty bidirectional ring in the bypass mode is not huge. The maximum difference of 40% is found for Water.

### 7.4.2 Execution time

A performance measure that is more meaningful from the system point of view is the program execution time. In Figure 7.13 the execution time of a system based on a faulty ring is normalized to the execution time of a system based on a good ring. Only the single ring system simulating the ring-down mode shows a significant increase in the execution time. The increase is significant for FFT, Radix, and Ocean; those benchmarks that show higher network utilization. The increase in execution time is as high as 37% for FFT, 16% for Radix, and 17% for Ocean. Both faulty rings that simulate the bypass mode show only a small increase in the execution time. The maximum increase of 6% is observed while running the Ray trace. Since the bidirectional ring becomes a single ring only when a node needs to communicate with the destination through the faulty node, for all other cases the ring remains the bidirectional ring. Due to locality, the volume of communication that is affected by the fault is low, hence we see that the execution time on a faulty ring simulating the bypass mode is lower than on a single ring.
Figure 7.7: Network utilization of good and faulty bidirectional rings for FFT

Figure 7.8: Network utilization of good and faulty bidirectional rings for Ocean
Figure 7.9: Network utilization of good and faulty bidirectional rings for Radix

Figure 7.10: Network utilization of good and faulty bidirectional rings for Ray trace
Figure 7.11: Network utilization of good and faulty bidirectional rings for Water-Nsquared

Figure 7.12: Network utilization of good and faulty bidirectional rings for Cholesky
Fault handling and reliability issues of the bidirectional ring were discussed in this chapter. A reliable implementation of the cache coherence protocol, described in Chapter 5, was also presented. The systems handling ring-down and bypass mode faults were simulated. The relative performance of good and faulty bidirectional rings were evaluated to measure the performance degradation caused by different types of faults. The performance degradation of benchmark applications on faulty bidirectional rings was found to be low, with the exception of a single ring that simulates the ring-down mode of failure. The ring handling bypass faults shows a small increase in the execution time of an application. The maximum increase of 6% was found for Ray trace. The two buffering schemes to handle bypass faults - separate and unified - show no significant difference in performance. The separate buffer scheme shows a nominal 2% lower execution time for Cholesky than the unified buffer scheme. This shows that either scheme can be selected, based on the simplicity of implementation rather than the expected performance penalty. In contrast, a single ring, which simulates the ring-down mode, shows the performance degradation as high as 37% for FFT. The network utilization of the rings handling bypass faults shows the same load balancing characteristic of a good bidirectional ring.

Figure 7.13: Normalized execution time for good and faulty bidirectional rings

Summary
This dissertation proposes the bidirectional ring as a viable alternative to the hierarchy of unidirectional rings for shared-memory multiprocessor systems. In addition to high performance, a major demand of scientific applications, reliability and cost-performance characteristic of an interconnection network have also become important issues due to emerging use of multiprocessors for commercial database applications. The simplicity of the bidirectional ring allows a high speed of operation and low cost of implementation. The bidirectional ring is found to give better performance and higher reliability than a hierarchy of unidirectional rings.

An invalidation-based cache coherence protocol is proposed for use with the bidirectional ring. The protocol uses the natural multicasting capability of rings. In this protocol a single invalidation packet is multicast to invalidate all copies of the cache line. The multicasting is achieved by adapting the routing scheme that is used for the hierarchy. The protocol guarantees sequential consistency. Unlike the hierarchy, two communication paths exist between any two nodes in the bidirectional ring, which allows an invalidation packet to outrun the response of a prior shared-read request. The invalidation can be delayed by introducing an invalidation pending state, which indicates that an invalidation is pending for the line that is to be received in response to a prior shared-read request. Upon receiving the line, the cache controller services the processor's read miss and then invalidates the line. An implementation scheme for the proposed protocol is presented that is capable of maintaining cache coherence despite a single node or link failure. The fault handling capability of the scheme is designed to be invoked only after detecting a fault, hence it does not affect the normal operation of the ring.

8.1 Performance

The dynamic clustering exhibited by a bidirectional ring is found to be effective in exploiting the communication locality of parallel applications to enhance their performance. The relative performance of the bidirectional ring and the hierarchy was evaluated by running two sets of
For the first set, a simple architecture was simulated, where each node has a single processor with a cache attached to it. We did not simulate a cache coherence protocol. By avoiding the complexity of the cache coherence protocol, this simulation could deal with system sizes up to 1K processors. A simple request-response transaction model was used, which supports single memory word and cache line transfers. Synthetic workloads were used to generate different memory access patterns, and different locality models were used to simulate spatial locality in the reference stream. The results show that the fixed cluster size of the hierarchy makes it very sensitive to the communication locality, especially for large networks. In contrast, the bidirectional ring shows a smooth improvement in performance as the communication locality is increased. The optimal locality for the hierarchy occurs when the communication is confined within clusters. Even for such workloads the performance of the bidirectional ring is comparable to the hierarchy. For low communication locality the bidirectional ring outperforms the hierarchy for all network sizes. For the workload showing no communication locality, the degradation in the performance of the bidirectional ring is much less pronounced than in the hierarchy.

The hotspot workload presents an interesting case. Although the bidirectional ring outperforms the hierarchy, the performance improvement is not as significant as in the case of other workload models. In the bidirectional ring, the receive buffer at the hotspot node overflows inducing a significant number of retries. This is less of a problem in the hierarchy, because the inter-ring interface buffers in the communication path share the load.

For the second set of simulations, a complete system based on the bidirectional ring using the proposed cache coherence protocol was simulated. Other system parameters are analogous to the NUMAchine prototype. This allows a direct comparison of a bidirectional ring-based system with that of the simulated NUMAchine prototype, which is based on the hierarchy. The relative performance was measured by running a set of applications from the SPLASH-2 benchmark suite. These benchmarks exhibit a variety of memory access patterns and communication to computation ratios. Their cache miss statistics also resemble the statistics for some commercial transaction processing benchmarks.

The bidirectional ring shows a significant reduction in latency, as much as 50% for the FFT benchmark. The latency is measured separately for exclusive read, shared read, and upgrade requests in processor clock cycles. The difference in latency is more pronounced for large cache line sizes, such as for 64 and 128 bytes, which increases both sharing and the volume of network traffic. In general, the bidirectional ring shows either comparable or better performance than the hierarchy. An interesting aspect of the bidirectional ring is that it handles bursty traffic having non-deterministic spatial distribution in a better way than the hierarchy. For example,
the communication pattern of Radix exhibits burstiness, which is also common in block transfers and prefetching. This is evident in comparing their ring access delays. The hierarchy shows high global ring access delays; especially for Radix it is as high as 210 ring cycles. In contrast, the ring access delays for the bidirectional ring are quite low, in the range of 2 to 20 ring cycles. These delays, however, are comparable with the access delays of the local rings of the hierarchy. In terms of overall performance, though no benchmark of the SPLASH-2 suite loads the network to the point where the execution time of the application is significantly affected, speedup curves show some gains for the bidirectional ring. Higher gains are observed for large line sizes.

8.2 Network Load Balancing

Another advantage of the bidirectional ring over the hierarchy is its tendency to balance the load over the two rings. Under certain conditions, such as low communication locality, the hierarchy may experience problems with the saturation of the high-level rings, whereas the low-level rings remain underutilized at the same time. In contrast, the bidirectional ring shows an essentially equal load on both rings. Our measure of network utilization, which is a more direct indicator of the network performance, shows no more than 5% difference in the utilization of the two rings of the bidirectional ring. The bidirectional ring has a tendency to balance the load over its constituent rings. In contrast, the hierarchy shows a significant difference in the utilization of its local and global rings. The utilization of the global ring is generally found to be twice as high as the utilization of the local rings.

8.3 Reliability

The reliability and fault handling issues of the bidirectional ring are also investigated in this dissertation. A reliable implementation of the cache coherence protocol is proposed, which is capable of handling node and link failures in the bypass mode. The systems handling faults in the ring-down and bypass modes were simulated to evaluate the relative performance of a good and a faulty bidirectional ring and measure the performance degradation caused by different types of simulated faults. The performance degradation of benchmark applications on faulty bidirectional rings is found to be low, with some exceptions on a single ring, which simulates the ring-down mode of failure. The single ring shows the performance degradation as high as 37% for FFT. The ring operating in the bypass mode shows a small increase in the execution time of an application. The maximum increase of 6% is found for Ray trace. The two buffering schemes to handle faults in the bypass mode - separate and unified - show no significant difference in
performance. For Cholesky, the separate-buffer scheme shows a nominal 2% reduction in the execution time in comparison with the unified buffer scheme. This shows that either scheme can be selected on the basis of the relative simplicity of implementation and cost. Another interesting feature of the faulty ring in the bypass mode is that it maintains the load balancing characteristic of a good bidirectional ring.

8.4 Future Work

8.4.1 Optical Interconnection

The bandwidth of electrical backplanes is limited by the physics of electronic and metal interconnect constraints. Transmission line effects, such as wave reflections and cross-talk, interference, electrical noise due to current changes and dielectric imperfections can all cause signal distortions and attenuation, clock skew, and random propagation delays. All these effects reduce the usable bandwidth in metal interconnects to a few Mbits/s [26, 35]. Free space optical backplanes can be used to achieve bandwidth of up-to Tbits/s over longer distance with lower error rate. The bidirectional ring can be embedded into the Hyperplane architecture proposed for implementing an intelligent optical backplane using Smart Pixel Arrays (SMA) [52]. The optical implementation of the bidirectional ring is simpler than the hierarchy, because there is no need to implement inter-ring interface buffers, which require optical-electrical-optical conversions. Jiang et al [22] have presented several routing schemes to deflect the packet that contends for an output link at an inter-ring interface. The deflection routing solves the optical-electrical-optical conversion problem by obviating the need for inter-ring interface buffers. But, deflected packets consume the bandwidth, which is otherwise available if the packets are buffered. The deflection routing may also cause break down of the NUMAchine cache coherence protocol [11].

8.4.2 Analytical Modeling

It would be useful to develop a good analytical model for the bidirectional ring so that the effects of different system parameters on its performance could be analyzed. Oi and Ranganathan [36] have developed models for the bidirectional and single unidirectional rings based on the M/G/1 queuing model. They have evaluated the relative performance of the two networks for up to 64 processors, and found the processor utilization is better on the bidirectional ring, particularly when the communication locality is high. They considered a bidirectional ring that has half the bandwidth of the unidirectional ring. This is not a useful comparison, because a single unidirectional ring does not exploit communication locality. The true competitor for the bidirectional ring is the hierarchy of unidirectional rings, because both have features to exploit
communication locality. The relative advantage of the bidirectional ring in comparison with the hierarchy is more pronounced for traffic patterns that show high burstiness and low spatial locality. Burstiness is a major cause of congestion at the inter-ring interface buffers of the hierarchy. Several analytical models for the hierarchy have been developed [30, 58], but none of them accurately models the network contention. Hamacher and Jiang introduce different parameters to model the contention in the hierarchy [16]. However, these parameters do not model the train effect (burstiness) in the traffic, which is a common cause of congestion at inter-ring interface buffers. It would be useful to develop a more complete analytical model of the bidirectional ring.

8.4.3 Bandwidth Reservation

Multiprocessors have been proposed for implementing multimedia parallel video servers [19, 41]. Video servers are responsible for storing and transporting video streams through a communication network to remote clients. A video server serves the requests of multiple clients for a number of videos simultaneously. Movies are divided into blocks and these blocks are stored across the nodes within the video server. Upon a client's request, movie blocks are collected from the storage nodes and synchronized for transmission as a video stream. The video server deals with the problem of allocating network resources to minimize the communication delays while forming video streams. Some sort of bandwidth reservation scheme is needed to get a bound on communication delays under network contention. A bandwidth reservation scheme for the bidirectional ring needs to be designed, which could support multiple concurrent transmission of video blocks.

8.4.4 Cache Coherence Protocol for Dynamic Clusters

The cache coherence protocol presented in this dissertation requires propagation of invalidation packets around the ring. It employs multicasting, which needs a single invalidation packet to traverse the ring. But, in this scheme, the invalidation packet transcends the dynamic clusters. With the exception of invalidation packets, the traffic caused by all other transactions is localized within the cluster which may give the benefit of localizing the effect of a fault to the faulty cluster. A protocol needs to be designed that limits the flow of an invalidation packet to its own cluster. One possibility is a simple directory protocol in which the home memory transmits invalidation packets separately to each invalidating cache, and waits for their acknowledgements. It then unlocks the home location and sends an acknowledgement to the requester. The relative performance of this scheme needs to be evaluated in comparison with the multicast-based scheme proposed in this dissertation.
Bibliography


